6.111 Fall 2017 Final Project Report

Emanuel Perez

Ziwen Jiang

Table of Contents

I. Introduction

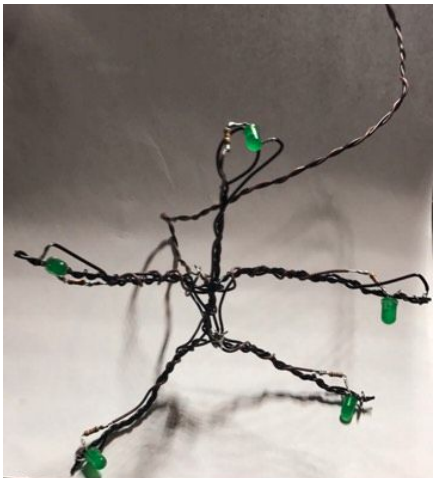
Virtual reality, such as VR and AR, has been a rising trend in the technology industry. Interactive gaming has indeed attracted great attention. Inspired by Ziwen's passion for virtual entertainment and procrastination, we decided to create our own interactive video game. We positioned several differently colored LEDs on selective parts of a 'body' (head, arms, and thighs) in order to track figure movement. This in turn would control a sprite on a VGA monitor. The sprite is be able to move in four directions, jump, kneel, and wave.

Our project can be split into two main functionalities: animation and motion processing. Under the animation aspect, there is game logic that appropriately updates the video display. With respect to the motion processing, there is object tracking for the LEDs.

II. Setup

The project required the use of an NTSC camera, differently colored LEDs (yellow and green but can also theoretically be done with red and blue), a VGA monitor, and the 6.111 labkit.

The original idea for the project was to position LEDs on a person, like how they were used for motion capture in *Avatar* and in the process of creating *NBA 2k15*. However, the LEDs that we worked with were not bright enough to be detected from a distance. Instead, Jacky created a 'body' out of a bunch of wires and soldered LEDs and resistors. The LEDs were positioned to simulate the head, arms, and legs. There was a common power and ground for the basic resistor-in-series-with-LED topology.



III. Modules Overview and Descriptions

The overall project can be split into 4 sections: motion capture, position detection, command converter, and graphics.

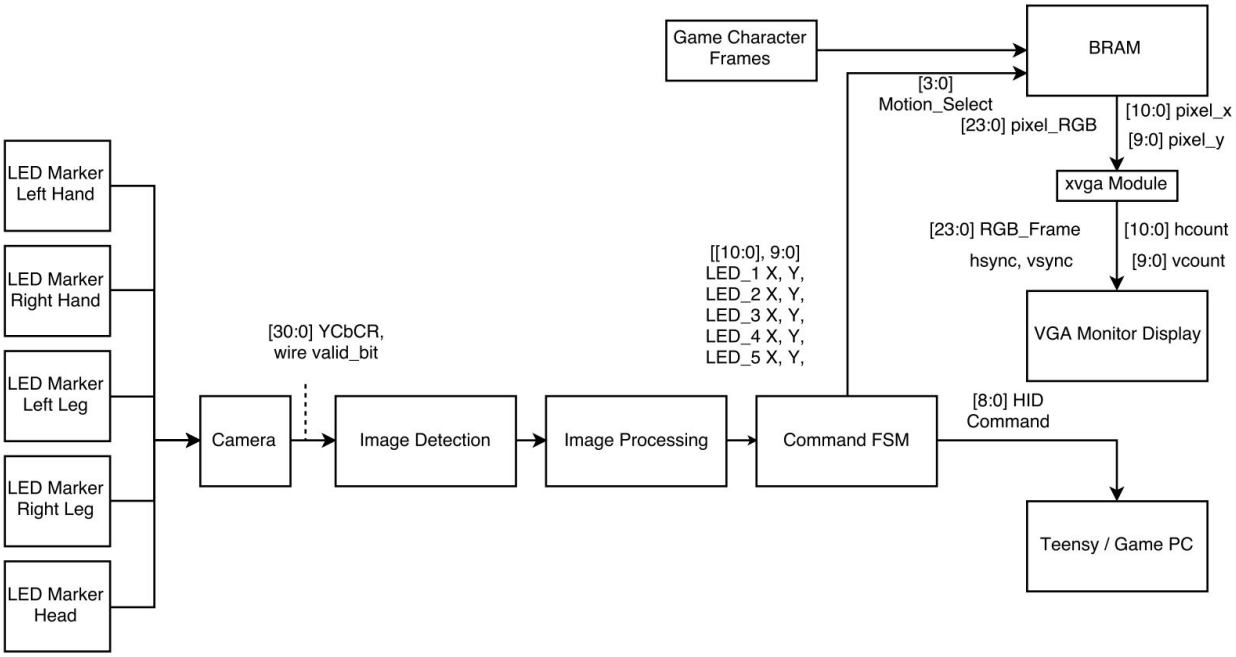


Figure 1. General Block Diagram

Motion Capture (Emanuel Perez)

This section of the project used code that was provided: ntsc_decode, ntsc_to_zbt, ycrcb2rgb. I created imageDetection and setThreshold.

ntsc_decode: This module takes streams of data from the NTSC video decoder and generates the corresponding pixels as a [29:0] YCrCb value. It also outputs f, v, h to communicate the frame and vertical and horizontal position that correspond to each pixel.

ntsc_to_zbt: This module prepares data and address values to fill ZBT memory with NTSC data.

ycrcb2rgb: This module converts [9:0] Y, Cr, Cb into [7:0] R,G,B values.

imageDetection: This module takes in the [7:0] R,G,B values from ycrcb2rgb and uses 7 bit threshold values for red, green, and blue to determine whether the pixel with the specified RGB value can be classified as red, green, yellow, or none of the above. The output was [1:0] pix, each of the four possibilities corresponding to a classification.

setThreshold: This module was created in order to troubleshoot and calibrate the threshold values for red, green, and blue. Using the switches and the enter button, you are able to switch a given threshold to a multiple of 8 (from 0 to 248 inclusive).
****

From the camera, we get a [29:0] YCrCb value which is then converted into RGB using ycrcb2rgb. To display a grayscale image, only the Y value of each pixel was saved using the ntsc_to_zbt module. However, in order to discriminate colors, I saved [1:0] pix, an output of imageDetection. I then used the [1:0] pix value in order to display a black image with blotches of yellow, red, and green corresponding to the lit LEDs.

To accomplish this, I created [7:0] pix_red, pix_green, pix_green and assigned them to vga_out_red, vga_out_green, vga_out_blue respectively. These values were changed at every positive edge of the clock cycle appropriately. For example, if pix was 0 (corresponding to a classification of neither red, green, or yellow) then pix_red, pix_green, and pix_blue would all be changed to 0 so that a black pixel would be displayed. Similar logic applies for when pix = 1, 2, and 3 (red, green, and yellow respectively).

The filter that I created was implemented by imageDetection. In order to determine the appropriate classification of a pixel given its RGB value, I had to empirically determine thresholds. For each each color classification red, green, and yellow there had to be 3

thresholds that had to be satisfied. For example, for green [7:0] G had to be greater than [7:0] g_th, [7:0] R had to be less than [7:0] r_tl, [7:0] B had to be less than [7:0] b_tl.
This filtered out ambient light in the lab since white light has high concentrations of R, G, and B. Similar thresholds were set for classifying red and yellow. The setThreshold module simplified the trial-and-error process that determined the appropriate threshold values.

Position Detection (Emanuel Perez)
I created the setAxis and centroidLocator modules for this part.

setAxis: This module was created in order to troubleshoot and calibrate the height and width of the regions that separate the video into 5 sections. Using the switches and the left and right buttons, you are able to move a boundary up/down or left/right as appropriate.

centroidLocator: This module used a naive algorithm to determine the centroid of each of the LEDs. Using the hcount and vcount, a unique area could be determined that corresponded to the region in which the pixel with coordinates hcount, vcount was located. For each region, a specific color was assigned. We decided to use green for all of our regions (since it was the simplest to work with) but this could be expanded to also include red and yellow. For each pixel, it checked what color it was and then kept a running sum of the first 32 hcounts and vcounts with that specified color (green in our case). At the end of each frame, it calculated the average of these samples and added it into an array. The centroid was then calculated as the average of the last 8 samples to account for noise. The reason that we used 32 and 8 for the the number of coordinates and samples respectively was so that we could capitalize on the fact that division by a power of two was just a simple bit shift to the right, thus simplifying calculations but this was at the expense of accuracy.

****

To simplify the process of calculating the centroid for each of the LEDs, the screen was divided into five sections which were divided like the following table. They were defined by three variables, one corresponding to the hcount corresponding to the vertical boundary and two vcounts corresponding to the two horizontal boundaries.

<table>
<tr><td colspan="2">1</td></tr>
<tr><td>2</td><td>3</td></tr>
<tr><td>4</td><td>5</td></tr>
</table>

The setAxis module allowed us to calibrate the boundaries for different distances from the camera and also to make sure that each of the LEDs were unique and separate in each region. Unfortunately, I was not able to get the centroidLocator module properly functioning. To debug, I displayed a blue screen blob on the screen with variable length with the centroid as its center. For the top LED (in region one), I was able to get the centroid bounce around to the left of its actual position. However, the deviations made it so that the calculated centroid was not accurate enough to pass on to the following modules for them to function properly.

Motion Command Module (Jacky Ziwen Jiang)

For command converter module, we created a command conversion FSM in FPGA to correlate the motion of different markers and specific command we want to send to control the game character. For the basic performance of our system, we will have six states: up, down, left, right, halt, and jump. For our reaching goal, we added additional movement, including left hand waving, right hand waving, and kneeling, which requires more complex coordinates tracking and comparison.

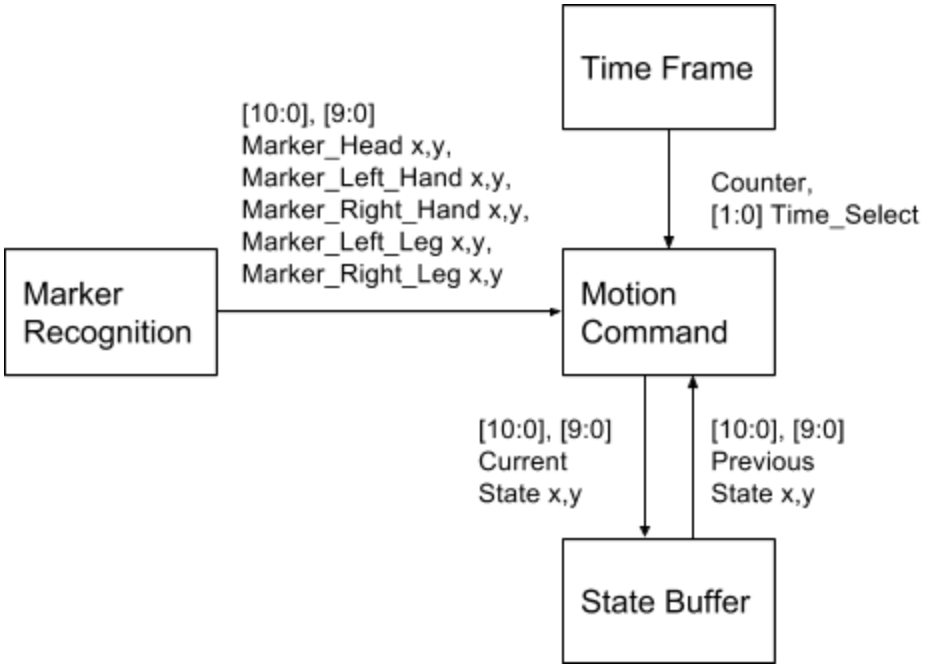


Figure . Motion Command Block Diagram

Marker Coordinates Tracking

In this section, we will introduce how the module will recognize the markers that corresponds to different body parts. The module takes the current coordinates of different markers as inputs, as well as storing the coordinates in buffer for comparison in the next section.

The original plan for the marker tracking was using the color of different LED that attach to different body parts. However, because the RGB color filter wasn't able to recognize color besides yellow and green, we decided to approach the problem in two different ways using relative positions of different markers.

The first approach is dividing the vga screen into 5 sections, while each section corresponds to one marker, as described in the Motion Recognition Module.

The second approach is directly taking the x, y coordinates, and RGB color of five markers, and using the relative positions of these five markers to determine which body part they corresponds to. The module first decided whether the person is facing front or back. If among all the marker inputs, there is one marker has RGB value that falls into the yellow range, we determined that the person is facing back, since the yellow LED marker is placed on the back side of the head. Otherwise, the person is

determined to face front. The marker with the highest y coordinate will correspond to head. When the person is facing front, the marker with the lowest x coordinate corresponds to left hand, because left hand has lower hcount when we converted the centroid; if facing back, the marker with the highest x coordinate corresponds to left hand, because the body orientation is reversed and left hand marker actually has the highest vcount. The same logic applied to the right hand marker. When facing front, the marker with the highest x coordinate corresponds to right hand. When facing back, the marker with lowest x coordinate is the right hand marker. The legs will have the lowest or second lowest y coordinates. When facing front, the one that has a lower x coordinate corresponds to the left leg, while the one that has a higher x coordinate corresponds to the right leg.

The module had two sets of coordinates of each body part for calculation. The first set of coordinates were the coordinates in the current VGA pixel frame. The second set of coordinates were the ones that were stored in the buffer certain time frame ago. We designed the module so that the time frame between the first set and second set of coordinates can be adjusted from 0.1 second to 0.5 second depending on how the actual system performed. The time frame is implemented with a counter.

Action Recognition

After we have the coordinate inputs of different markers, the module used the trajectory of each marker to determine the specific movement of the person.
First, the module will receive the signal from the previous section about the facing direction of the person. After that, the module will calculate the trajectory of each marker and relative positions between different markers to determine the actual action of the body. If the trajectory distance exceeds certain threshold, the action will be determined. Here is the table of how we determined specific actions:

Action	Determination Method	Default Threshold
Up	While facing front, if kneel_threshold > left_leg_y - left_leg_y_buffer > up_threshold,	Up_threshold = 50; Kneel_threshold = 100

	then action is up	
Down	While facing back, if kneel_threshold > left_leg_y - left_leg_y_buffer > up_threshold, then action is up	Down_threshold = 50; Kneel_threshold = 100
Left	While facing front, if left_leg_x_buffer - left_leg_x > left_threshold, then action is left; While facing back, if left_leg_x- left_leg_x_buffer > left_threshold, then action is left;	Left_threhold = 50
Right	While facing front, if right_leg_x - right_leg_x > right_threshold, then action is right; While facing back, if right_leg_x_buffer- right_leg_x > right_threshold, then action is right;	Right_threhold = 50
Jump	If head_y - head_y_buffer > jump_threshold, then action is jump	Jump_threshold = 100
Left_hand_wave	If left_hand_y_buffer - left_hand_x > left_hand_wave_threshold, then action is left_hand_wave;	Left_hand_wave_threshold = 50
Right_hand_wave	If right_hand_y_buffer - right_hand_x > right_hand_wave_threshold, then action is right_hand_wave;	Right_hand_wave_threshold = 50
Kneel	If (head_y_buffer - left_leg_y_buffer) - (head_y - left_leg_y)> up_threshold, then action is kneel	Kneel_threshold = 100

In the chart above, the x, y coordinates without “buffer” in the end are the coordinates of each marker in the current frame, while the ones with “buffer” are the coordinates that stored in the buffer certain time frame ago. The thresholds of each

section can be changed internally, while the actual values were supposed to be determined by the actual measurement and testings.

When no specific motion is detected, the module will generate “Halt” action, which is the steady mode.

Each action is encoded into a 4-bit code that is transferred into the display module

Display Module (Jacky Ziwen Jiang)

After the motion code is created and transferred, the display module have two types of output. The first output is direct graphics change on the monitor that the FPGA connects to. For this output, we created a simple game character and display it on the monitor, while the motion command module will control the movement of the character. The second output will be a HID protocol that correspond to keyboard presses and mouse click. The output will be used to control character in a FPS game on a separate computer, so that the movement of the character will match our own detected motions.

VGA Display

The game character is generated by a pixel module inspired by Weston’s implementation.

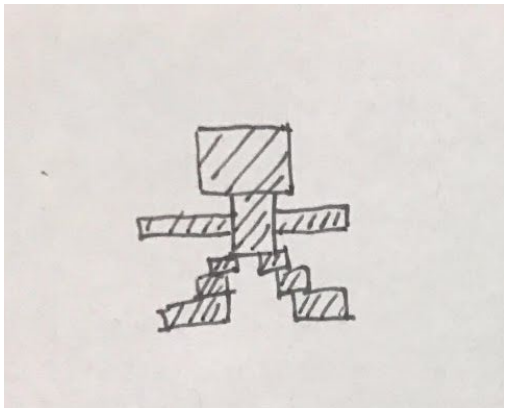
Inside of the VGA display module, there are internal x, y coordinates stored in registers that keep track of the positions of the game character. The size of the character can also be controlled by a 4-bit register that corresponds to 3 different sizes (1x, 2x, 4x, 8x).

To generate the character, we first created a 9 x 10 block of pixels, and fill in the RGB color for pixels within the block line by line. The 4-bit size of the character that is stored in the size register will be used as number of bit shift for the pixel block.

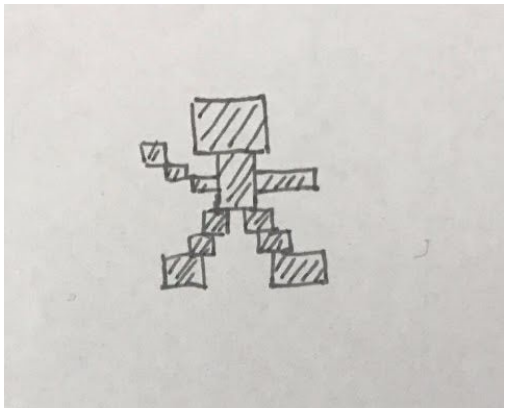
For implementation, we first right shift number of size bits for vcount and hcount. If vcount and hcount are still in the the pixel block, we will then output the RGB color

that corresponds to the location. For each action, a specific arrangement of pixel block is created in verilog.

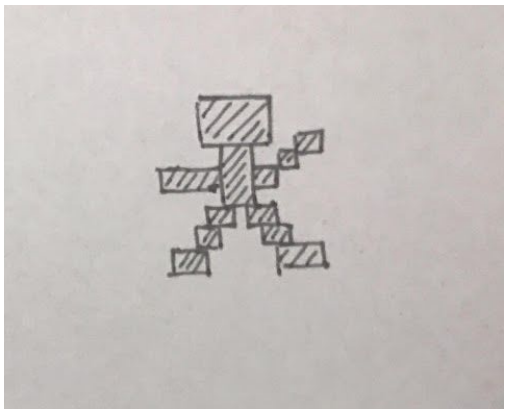
The following pictures corresponding to the shape of the game character in different actions:



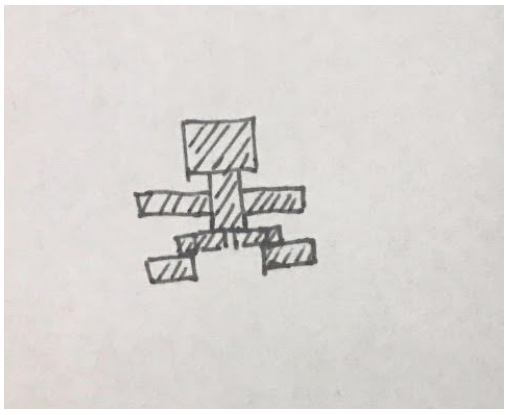
Halt



Left Hand Wave



Right Hand Wave



Kneel

Figure 2. VGA Game Character Display

For up, down, left, and right motion, the x, y coordinates of the game character changed according to the times of motion commend that is sent. Each motion command corresponds to 50 pixel change, which can be changed through internal Verilog code.

For left hand wave, right hand wave, and kneeling, the character actions are shown above.

For each jump command, we change the size of the character from 16x to 1x, and reverse it back to 16x to create jump action. The time frame between each size

change is controlled by another counter, which can be changed depending on the visual requirement.

After each action command, the character will always go back to halt state and waits for the next command.

HID Control

For a more immersive gaming experience, we wanted to use our motion sensing system to control a game character in an actual FPS game. To transfer the action on FPGA to computer through HID protocol, we used a Teensy 3.2 as a bridge for signal transfer.

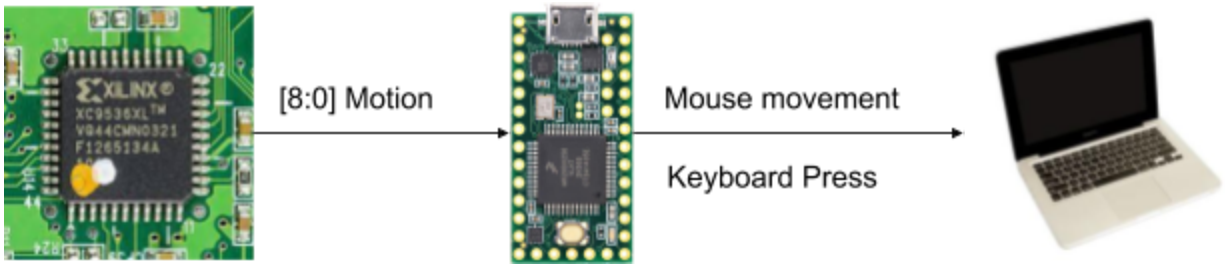


Figure 3. HID Module Signal Diagram

Teensy 3.2 can be recognized as a HID device by computer, while it has built-in library that allows user to write keyboard press and mouse movement command directly. We used a 9-bit register to store all the action information, in which each bit corresponds to a specific action, and output the command through User1 GPIO from FPGA kit. Each GPIO pin was then connected to input digital pin on Teensy. We pulldown the digital pin on Teensy. Everytime the digital pins on Teensy received a “1” from FPGA, it will output specific HID action that is transferred to computer. A specific correlation diagram between motion and HID signal is shown below:

Motion	Bit Number (FPGA)	Pin Number (Teensy)	HID Action

Up	0	D0	Keypress_W
Down	1	D1	Keypress_S
Left	2	D2	Keypress_A
Right	3	D3	Keypress_D
Jump	4	D4	Keypress_Space
Left Hand Wave	5	D5	Mouse_Left / Keypress_X
Right Hand Wave	6	D6	Mouse_Right/ Keypress_Y
Kneel	7	D7	Mouse_Down/ Keypress_Z
Halt	8	D8	Halt

As a result, although we weren't able to convert the physical motion into digital command signal, we controlled a simple FPS game using the push button on FPGA kit as the motion command.

IV. Lessons Learned

Color thresholding was a lot more buggy than was expected. Playing around with the threshold values to get the appropriate LED and its color to display was annoying and was partially aided by the module that allowed you to set the threshold values. Green and yellow were more easily detected than red after much testing. As such creating modules that allow you to more easily change values (instead of changing them in verilog and then having to reupload everything) saves more time and headache the earlier and more robust they are created.

For marker detection, we encountered certain trouble of calculating the centroids of each marker. We were able to separate the screen into different sections, as well as apply RGB filter to detect marker color. However, when we tried to calculate the centroids, the result centroid became really unstable, partially because we didn't really

implement a good averaging method. At the same time, we should learn to develop a better testing and validation methods since we weren't able to test which module went wrong that caused the centroid error.

Overall, a lesson we learned is to finish each individual part early so that we would have more time to work on the integration between different modules.

V. Future Work

In the future, there are couple things we can work on to improve our project:

1. Debugging and developing a better centroid calculation method for more stable and accurate centroid results;
2. Connection between the position detection module and motion command module.
3. Use infrared LEDs instead of different colored LEDs since there would be much less noise to deal with

VI. Appendix: Verilog Code

xvga.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:        18:42:37 11/14/2017
// Design Name:
// Module Name:      xvga
// Project Name:
// Target Devices:
// Tool versions:
```

```
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////
//
// xvga: Generate XVGA display signals (1024 x 768 @ 60Hz)
//
//////////////////////////////////////////////////////////////////////////////////

module xvga(input vclock,
            output reg [10:0] hcount,    // pixel number on current line
            output reg [9:0] vcount,     // line number
            output reg vsync,hsync,blank);

  // horizontal: 1344 pixels total
  // display 1024 pixels per line
  reg hblank,vblank;
  wire hsyncon,hsyncoff,hreset,hblankon;
  assign hblankon = (hcount == 1023);
  assign hsyncon = (hcount == 1047);
  assign hsyncoff = (hcount == 1183);
  assign hreset = (hcount == 1343);
```

```
  // vertical: 806 lines total
  // display 768 lines
  wire vsyncon,vsyncoff,vreset,vblankon;
  assign vblankon = hreset & (vcount == 767);
  assign vsyncon = hreset & (vcount == 776);
  assign vsyncoff = hreset & (vcount == 782);
  assign vreset = hreset & (vcount == 805);

  // sync and blanking
  wire next_hblank,next_vblank;
  assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
  assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;
  always @(posedge vclock) begin
      hcount <= hreset ? 0 : hcount + 1;
      hblank <= next_hblank;
      hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync;  // active low

      vcount <= hreset ? (vreset ? 0 : vcount + 1) : vcount;
      vblank <= next_vblank;
      vsync <= vsyncon ? 0 : vsyncoff ? 1 : vsync;  // active low

      blank <= next_vblank | (next_hblank & ~hreset);
  end
endmodule
```

teensy code

```
void setup() {
```

```
  // put your setup code here, to run once:
  //keyboard
  pinMode(0, INPUT_PULLDOWN); // UP
  pinMode(1, INPUT_PULLDOWN); // DOWN
  pinMode(2, INPUT_PULLDOWN); // LEFT
  pinMode(3, INPUT_PULLDOWN); // RIGHT
  pinMode(4, INPUT_PULLDOWN); // JUMP
  pinMode(5, INPUT_PULLDOWN); // LEFT_HAND_WAVE
  pinMode(6, INPUT_PULLDOWN); // RIGHT_HAND_WAVE
  pinMode(7, INPUT_PULLDOWN); // KNEEL
  //mouse part
  pinMode(8, INPUT_PULLDOWN); // move left
  pinMode(9, INPUT_PULLDOWN); // move right
  pinMode(10, INPUT_PULLDOWN); // move up
  pinMode(11, INPUT_PULLDOWN); // move down
}

void loop() {
  if (digitalRead(0) == LOW) {
        Keyboard.press(KEY_W);
        }
  else if (digitalRead(0) == HIGH) {
        Keyboard.release(KEY_W);
        }
  if (digitalRead(1) == LOW) {
        Keyboard.press(KEY_S);
        }
  else if (digitalRead(1) == HIGH) {
        Keyboard.release(KEY_S);
```

```
}

if (digitalRead(2) == LOW) {
        Keyboard.press(KEY_A);

        }
else if (digitalRead(2) == HIGH) {
        Keyboard.release(KEY_A);
}

if (digitalRead(3) == LOW) {
        Keyboard.press(KEY_D);

        }
else if (digitalRead(3) == HIGH) {
        Keyboard.release(KEY_D);
}

if (digitalRead(4) == LOW) {
        Keyboard.press(KEY_SPACE);

        }
else if (digitalRead(4) == HIGH) {
        Keyboard.release(KEY_SPACE);
}

if (digitalRead(5) == LOW) {
        Keyboard.press(KEY_X);
```

```
        }
else if (digitalRead(5) == HIGH) {
        Keyboard.release(KEY_X);
}

if (digitalRead(6) == LOW) {
        Keyboard.press(KEY_Y);

 }

else if (digitalRead(6) == HIGH) {
        Keyboard.release(KEY_Y);
}

if (digitalRead(7) == LOW) {
        Keyboard.press(KEY_Z);

        }
else if (digitalRead(7) == HIGH) {
        Keyboard.release(KEY_Z);

}

//mouse
if (digitalRead(8) == HIGH) { //left
        Mouse.move(-2,0);
        delay(1);
        }
if (digitalRead(9) ==  HIGH) { //right
```

```
        Mouse.move(2,0);
        delay(1);
        }
  if (digitalRead(10) ==  HIGH) { //up
        Mouse.move(0,2);
        delay(1);
        }
  if (digitalRead(11) ==  HIGH) { //down
        Mouse.move(0,-2);
        delay(1);
        }

}
```

Motion_Recognization.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:        21:22:15 12/04/2017
// Design Name:
// Module Name:      marker_recog
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
```

```
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module motion_recognization #(parameter time_frame=2700000,//create a 0.1 delay
between
                                                    parameter jump_threshold =
200, //the displacement in pixel avlue for jump threshold
                                                    parameter up_threshold = 200,
                                                    parameter down_threshold =
200,
                                                    parameter left_threshold = 200,
                                                    parameter right_threshold =
200,
                                                    parameter right_wave_threshold
= 200,
                                                    parameter left_wave_threshold
= 200,
                                                    parameter kneel_threshold =
200)
(
    input clk,
    input reset,
    input [9:0] led_1_x, //head led
    input [10:0] led_1_y,
    input [9:0] led_2_x, //body led
```

```
input [10:0] led_2_y,
input [9:0] led_3_x, //left_arm_led
input [10:0] led_3_y,
input [9:0] led_4_x, //right_arm_led
input [10:0] led_4_y,
input [9:0] led_5_x, //left_leg_led
input [10:0] led_5_y,
input [9:0] led_6_x, //right_leg_led
input [10:0] led_6_y,
input front_back, // the facing directions of the body
input left,
input right,
input [1:0] reading_frame,
output reg [3:0] motion
    );
// the buffer register that store x and y coordinates in the previous timeframe
reg [9:0] buffer_1_x;
reg [9:0] buffer_1_y;
reg [9:0] buffer_2_x;
reg [9:0] buffer_2_y;
reg [9:0] buffer_3_x;
reg [9:0] buffer_3_y;
reg [9:0] buffer_4_x;
reg [9:0] buffer_4_y;
reg [9:0] buffer_5_x;
reg [9:0] buffer_5_y;
reg [9:0] buffer_6_x;
reg [9:0] buffer_6_y;
reg [19:0] reading_window;
```

```
    reg [19:0] counter; // the counter is for reading time frame
    parameter reading_window = time_frame*reading_frame;

    always @(posedge clk) begin
        if (reset) begin
            counter <= 20'd0;
        end
        else if (counter == reading_window) begin // create a certain time delay between
the initial and end
            if (front_back == 1) begin
                    // reading to determine the trajectory
                if ((led_1_y > buffer_1_y)&&(led_1_y - buffer_1_y >
jump_threshold)) begin //if the displacement is higher than the threshold,

                                    // then the module will output motion as jump
                    motion <= 4'd4; // jump motion code
                end
                else if ((led_5_y > buffer_5_y) && (led_5_y - buffer_5_y >
up_threshold)) begin //facing left, so that the right leg led will be viewed in camera
                    motion <= 4'b1001; //up motion
                end
                else if ((led_5_y > buffer_5_y) && (led_5_y - buffer_5_y >
up_threshold)) begin //facing left, so that the right leg led will be viewed in camera
                    motion <= 4'b0001; //up motion
                end
                else if ((buffer_3_y > led_3_y )&&(buffer_3_y - led_3_y) >
left_wave_threshold) begin
                    motion <= 4'd5; // left wave motion
                end
```

```
                else if ((led_3_y > buffer_3_y )&&(led_3_y - buffer_3_y ) >
left_wave_threshold) begin
                        motion <= 4'd7; // put left hands down
                end

                else if ((led_4_y > buffer_4_y)&&(led_4_y - buffer_4_y) >
right_wave_threshold) begin
                        motion <= 4'd6; // right wave motion
                end
                else if ((buffer_4_y > led_4_y )&&(buffer_4_y-led_4_y) >
right_wave_threshold) begin
                        motion <= 4'd7; // put right hands down
                end
                else if ((led_5_x < buffer_5_x) && (buffer_5_x - led_5_x>
left_threshold)) begin //facing left,  left leg led will be viewed in camera
                        motion <= 4'd2; //left motion
                end
                else if ((led_6_x > buffer_6_x) && (led_6_x - buffer_6_x >
right_threshold)) begin //facing right,  right leg led will be viewed in camera
                        motion <= 4'd3; //right motion
                end
                else if ((led_1_y - led_5_y) - (buffer_1_y - buffer_5_y)>
kneel_threshold) begin
                        motion <= 4'd8; //kneel motion
                end
                else begin
                        motion <= 4'd7; //halt
                end
```

```
                end
                else if (front_back == 0) begin //facing back
                        if ((led_1_y > buffer_1_y)&&(led_1_y - buffer_1_y >
jump_threshold)) begin //if the displacement is higher than the threshold,

                                                // then the module will output motion as jump
                                motion <= 4'd4; // jump motion code
                        end
                        if ((led_5_y > buffer_5_y) && (led_5_y - buffer_5_y >
down_threshold)) begin // the movement is down now
                                motion <= 4'd1; //down motion
                        end
                        if ((buffer_3_y > led_3_y )&&(buffer_3_y - led_3_y) >
left_wave_threshold) begin // since we are facing back now, the left hand is actually


                 // appearing on the right
                                motion <= 4'd5; // left wave motion on the screen, which
corresponds to right hand wave when facing back
                        end
                        else if ((led_3_y > buffer_3_y )&&(led_3_y - buffer_3_y ) >
left_wave_threshold) begin
                                motion <= 4'd7; // put left hands down
                        end
                        if ((led_4_y > buffer_4_y)&&(led_4_y - buffer_4_y) >
right_wave_threshold) begin
                                motion <= 4'd6; // right wave motion
                        end
```

```
                else if ((buffer_4_y > led_4_y )&&(buffer_4_y-led_4_y) >
right_wave_threshold) begin
                        motion <= 4'd7; // put right hands down
                end

                if ((led_5_x < buffer_5_x) && (buffer_5_x - led_5_x>
left_threshold)) begin //facing left,  left leg led will be viewed in camera
                        motion <= 4'd2; //left motion
                end
                if ((led_6_x > buffer_6_x) && (led_6_x - buffer_6_x >
right_threshold)) begin //facing right,  right leg led will be viewed in camera
                        motion <= 4'd3; //right motion
                end
                if ((led_1_y - led_5_y) - (buffer_1_y - buffer_5_y)>
kneel_threshold) begin
                        motion <= 4'd8; //kneel motion
                end
        end

        //renew the buffer with current reading
        buffer_1_x <= led_1_x;
        buffer_1_y <= led_1_y;
        buffer_2_x <= led_2_x;
        buffer_2_y <= led_2_y;
        buffer_3_x <= led_3_x;
        buffer_3_y <= led_3_y;
        buffer_4_x <= led_4_x;
        buffer_4_y <= led_4_y;
        buffer_5_x <= led_5_x;
```

```
                buffer_5_y <= led_5_y;
                buffer_6_x <= led_6_x;
                buffer_6_y <= led_6_y;

                counter <= 20'd0; //reset counter
        end
        else begin
                counter <= counter +1;
        end
    end




endmodule
```

Motion_Command.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:         19:28:35 11/20/2017
// Design Name:
// Module Name:      puck_conversion_two
// Project Name:
// Target Devices:
// Tool versions:
// Description:
```

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments: ools /mit/6.1/tools.tcsh

//
//////////////////////////////////////////////////////////////////////////////////
module Motion_Command(
    input clk,
    input [10:0] hcount, // syncronize the conversion module with frame rate of VGA
    input [9:0] vcount,
    input [3:0] motion,
    input up,
    input down,
    input left,
    input right,
    input jump,
    input left_wave,
    input right_wave,
    input kneel,
    input reset,
    input left_mouse,
    input right_mouse,
    input up_mouse,
    input down_mouse,
    output [23:0] p_out,
    output reg [7:0] keyboard,
```

```
output reg [3:0] mouse,
output reg [7:0] led
      );
//initilize
parameter BLACK = 24'h00_00_00;
parameter RED = 24'hff_00_00;
parameter TAN = 24'hde_b8_87;
parameter BROWN = 24'h99_66_33;
parameter CLEAR = 24'h00_00_00;
// parameter SCALE = 4; //power of two scaling factor
parameter HEIGHT = 10;
parameter WIDTH = 9;

// action state parameters
parameter up_state = 4'b1001;
parameter down_state = 4'b0001;
parameter left_state = 4'b0010;
parameter right_state = 4'b0011;
parameter jump_state = 4'b0100;
parameter left_wave_state = 4'b0101;
parameter right_wave_state = 4'b0110;
parameter halt_state = 4'b0111;
parameter kneel_state = 4'b1000;

reg [23:0] pixel;
reg [7:0] location;
reg [10:0] current_x;
reg [9:0] current_y;
reg [1:0] current_scale;
```

```
    reg jump_loop; //this register is to make sure that the jump loop only happens once
for every jump input
    reg [3:0] state;
    // assign scale = current_scale;
    assign p_out = pixel;
    reg up_flag;
    reg [4:0] jump_counter;

    always @ (posedge clk) begin
        // testing purpose
        /*
            4'b0000;
            4'b0001;
            4'b0010;
            4'b0011;
            4'b0100;
            4'b0101;
            4'b0110;
            4'b0111;
        */
        //deermine state and keyboard press output

        if (up || motion == up_state) begin
            state <= 4'b1001;
            keyboard <= 8'b11111110;
            led <= 8'b00000001; // led marker to indicate the action detected
        end
        else if (down || motion == down_state) begin
            state <= 4'b0001;
```

```
                keyboard <= 8'b11111101;
                led <= 8'b00000010;
        end
        else if (left || motion == left_state) begin
                state <=4'b0010;
                keyboard <= 8'b11111011;
        ools /mit/6.1/tools.tcsh
led <= 8'b00000100;
        end
        else if (right || motion == right_state) begin
                state <=4'b0011;
                keyboard <= 8'b11110111;
                led <= 8'b00001000;
        end
        else if (jump|| motion == jump_state) begin
                state <=4'b0100;
                keyboard <= 8'b11101111;
                led <= 8'b00010000;
        end
        else if (left_wave|| motion == left_wave_state) begin
                state <=4'b0101;
                // keyboard <= 8'b11011111;
                mouse <= 4'b0001; //we will use left wave for moving mouse right
                led <= 8'b00100000;
        end
        else if (right_wave|| motion == right_wave_state) begin
                state <=4'b0110;
                // keyboard <= 8'b10111111;
                mouse <= 4'b0010; // we will use right wave for moving mouse left
```

```
        led <= 8'b01000000;
end
else if (kneel|| motion == kneel_state) begin
        state <= 4'b1000;
        // keyboard <= 8'b01111111;
        mouse <= 4'b1000; // move mouse down
        led <= 8'b10000000;
end
else if (motion == halt_state) begin// halt state
        state <=4'b0111;
        keyboard <= 8'b11111111;
        mouse <= 4'b0000;
        led <= 8'b00000000;
end
else begin
        state <=4'b0111; ools /mit/6.1/tools.tcsh

        keyboard <= 8'b11111111;
        mouse <= 4'b0000;
        led <= 8'b00000000;
end

// gun/mouse movement
/*
if (left_mouse)
        mouse <= 4'b0001;
else if (right_mouse)
        mouse <= 4'b0010;
else if (up_mouse)
```

```
        mouse <= 4'b0100;
else if (down_mouse)
        mouse <= 4'b1000;
else
        mouse <= 4'b0000;
*/
//VGA display section

if (hcount == 0&& vcount == 0) begin

        if (reset) begin
                current_x <= 10'd512;
                current_y <= 9'd384;
                current_scale <= 2'd3;
                state <= halt_state;
        end

        else if (state == up_state) begin
                if (current_y < 10)
                        current_y <= 0;
                else
                        current_y <= current_y - 10;
                state <= halt_state;
        end

        else if (state == down_state) begin
                if (current_y > 680)
                        current_y <= 681;
                else
```

```
                current_y <= current_y + 10;
            state <= halt_state;
        end

        else if (state == left_state) begin
            if (current_x < 10)
                current_x <= 0;
            else
                current_x <= current_x - 10;
            state <= halt_state;
        end

        else if (state == right_state) begin // because of the scaling factor, we want to show the whoe figure, so that we will reserve more area
                                                                    // on right
            if (current_x >952) begin
                current_x <= 953;
                state <= halt_state;
            end
            else begin
                current_x <= current_x + 10;
                state <= halt_state;
            end
        end

        else if (state == jump_state) begin // for the jump out put, we will change on the scaling factor
                if (jump_counter < 5'd18) begin
                    if (up_flag) begin
```

```
                        jump_counter <= jump_counter + 1;
                        if (jump_counter == 5'd3)
                                current_scale <= current_scale - 1;

                        if (jump_counter == 5'd6)
                                current_scale <= current_scale - 1;
                        if (jump_counter == 5'd9)
                                current_scale <= current_scale - 1;
                        if (current_scale == 2'd3)
                                up_flag <= 0;
                end
                else if (~up_flag) begin
                        jump_counter <= jump_counter + 1;
                        if (jump_counter == 5'd12)
                                current_scale <= current_scale + 1;

                        if (jump_counter == 5'd15)
                                current_scale <= current_scale + 1;
                        if (jump_counter == 5'd17)
                                current_scale <= current_scale + 1;
                end
        end
        else if (jump_counter >= 5'd18) begin
                jump_counter <= 0;
                current_scale <= 2'd3;
                state <= halt_state; // one loop has been finished
        end
end
```

```
                // else state <= halt_state;
        end
        // puck display section
        if (((hcount-1) >= current_x && (hcount-1) <
(current_x+(WIDTH<<current_scale))) && ((vcount + 1) >= current_y && (vcount+ 1) <
(current_y+(HEIGHT<<current_scale)))) begin

        //test
        // if ((hcount >= 100 && hcount < 200) && (vcount >= 100 && vcount < 200))
begin
                location <= ((hcount-current_x)>>current_scale) + (((vcount -
current_y))>>current_scale)*WIDTH;
                //test
                //location <= hcount>>1;
                if (state == halt_state || state == up_state || state == left_state || state ==
right_state || state == down_state || state == jump_state ) begin
                        case(location)
                        0: pixel <= BLACK; //ROW 1
                        1: pixel <= BLACK;
                        2: pixel <= RED;
                        3: pixel <= RED;
                        4: pixel <= RED;
                        5: pixel <= RED;
                        6: pixel <= RED;
                        7: pixel <= BLACK;
                        8: pixel <= BLACK;

                        9: pixel <= BLACK; //ROW 2
                        10: pixel <= BLACK;
```

```
11: pixel <= RED;
12: pixel <= RED;
13: pixel <= RED;
14: pixel <= RED;
15: pixel <= RED;
16: pixel <= BLACK;
17: pixel <= BLACK;

18: pixel <= BLACK; //3 ROW
19: pixel <= BLACK;
20: pixel <= RED;
21: pixel <= RED;
22: pixel <= RED;
23: pixel <= RED;
24: pixel <= RED;
25: pixel <= BLACK;
26: pixel <= BLACK;

27: pixel <= BLACK;//4 ROW
28: pixel <= BLACK;
29: pixel <= BLACK;
30: pixel <= RED;
31: pixel <= RED;
32: pixel <= RED;
33: pixel <= BLACK;
34: pixel <= BLACK;
35: pixel <= BLACK;

36: pixel <= BLACK;//5 ROW
```

```
37: pixel <= BLACK;
38: pixel <= BLACK;
39: pixel <= RED;
40: pixel <= RED;
41: pixel <= RED;
42: pixel <= BLACK;
43: pixel <= BLACK;
44: pixel <= BLACK;

45:pixel <= RED; //6 ROW
46:pixel <= RED;
47:pixel <= RED;
48:pixel <= RED;
49:pixel <= RED;
50:pixel <= RED;
51:pixel <= RED;
52:pixel <= RED;
53:pixel <= RED;

54: pixel <= BLACK;//7 ROW
55: pixel <= BLACK;
56: pixel <= BLACK;
57: pixel <= RED;
58: pixel <= RED;
59: pixel <= RED;
60: pixel <= BLACK;
61: pixel <= BLACK;
62: pixel <= BLACK;
```

```
63: pixel <= BLACK; //8 ROW
64: pixel <= BLACK;
65: pixel <= BLACK;
66: pixel <= RED;
67: pixel <= BLACK;
68: pixel <= RED;
69: pixel <= BLACK;
70: pixel <= BLACK;
71: pixel <= BLACK;

72: pixel <= BLACK; //9 ROW
73: pixel <= BLACK;
74: pixel <= RED;
75: pixel <= BLACK;
76: pixel <= BLACK;
77: pixel <= BLACK;
78: pixel <= RED;
79: pixel <= BLACK;
80: pixel <= BLACK;


81: pixel <= RED; //10 ROW
82: pixel <= RED;
83: pixel <= BLACK;
84: pixel <= BLACK;
85: pixel <= BLACK;
86: pixel <= BLACK;
87: pixel <= BLACK;
88: pixel <= RED;
```

```
            89: pixel <= RED;

            default: pixel <= BLACK;
            endcase
    end
    else if (state == left_wave_state) begin
            case(location)
            0: pixel <= BLACK; //ROW 1 head
            1: pixel <= BLACK;
            2: pixel <= RED;
            3: pixel <= RED;
            4: pixel <= RED;
            5: pixel <= RED;
            6: pixel <= RED;
            7: pixel <= BLACK;
            8: pixel <= BLACK;

            9: pixel <= BLACK; //ROW 2
            10: pixel <= BLACK;
            11: pixel <= RED;
            12: pixel <= RED;
            13: pixel <= RED;
            14: pixel <= RED;
            15: pixel <= RED;
            16: pixel <= BLACK;
            17: pixel <= BLACK;

            18: pixel <= BLACK; //3 ROW
            19: pixel <= BLACK;
```

```
20: pixel <= RED;
21: pixel <= RED;
22: pixel <= RED;

23: pixel <= RED;
24: pixel <= RED;
25: pixel <= BLACK;
26: pixel <= BLACK;

27: pixel <= RED;//4 ROW neck
28: pixel <= BLACK;
29: pixel <= BLACK;
30: pixel <= RED;
31: pixel <= RED;
32: pixel <= RED;
33: pixel <= BLACK;
34: pixel <= BLACK;
35: pixel <= BLACK;

36: pixel <= BLACK;//5 ROW
37: pixel <= RED;
38: pixel <= BLACK;
39: pixel <= RED;
40: pixel <= RED;
41: pixel <= RED;
42: pixel <= BLACK;
43: pixel <= BLACK;
44: pixel <= BLACK;
```

```
45:pixel <= BLACK; //6 ROW hand
46:pixel <= BLACK;
47:pixel <= RED;
48:pixel <= RED;
49:pixel <= RED;
50:pixel <= RED;
51:pixel <= RED;
52:pixel <= RED;
53:pixel <= RED;

54: pixel <= BLACK;//7 ROW     WAIST
55: pixel <= BLACK;
56: pixel <= BLACK;
57: pixel <= RED;
58: pixel <= RED;
59: pixel <= RED;
60: pixel <= BLACK;
61: pixel <= BLACK;
62: pixel <= BLACK;

63: pixel <= BLACK; //8 ROW LEG
64: pixel <= BLACK;
65: pixel <= BLACK;
66: pixel <= RED;
67: pixel <= BLACK;
68: pixel <= RED;
69: pixel <= BLACK;
70: pixel <= BLACK;
71: pixel <= BLACK;
```

```
        72: pixel <= BLACK; //9 ROW
        73: pixel <= BLACK;
        74: pixel <= RED;
        75: pixel <= BLACK;
        76: pixel <= BLACK;
        77: pixel <= BLACK;
        78: pixel <= RED;
        79: pixel <= BLACK;
        80: pixel <= BLACK;

        81: pixel <= RED; //10 ROW
        82: pixel <= RED;
        83: pixel <= BLACK;
        84: pixel <= BLACK;
        85: pixel <= BLACK;
        86: pixel <= BLACK;
        87: pixel <= BLACK;
        88: pixel <= RED;
        89: pixel <= RED;
        default: pixel <= BLACK;
        endcase
end
else if (state == right_wave_state) begin
case(location)
        0: pixel <= BLACK; //ROW 1
        1: pixel <= BLACK;
        2: pixel <= RED;
        3: pixel <= RED;
```

```
4: pixel <= RED;
5: pixel <= RED;
6: pixel <= RED;
7: pixel <= BLACK;
8: pixel <= BLACK;

9: pixel <= BLACK; //ROW 2
10: pixel <= BLACK;
11: pixel <= RED;
12: pixel <= RED;
13: pixel <= RED;
14: pixel <= RED;
15: pixel <= RED;
16: pixel <= BLACK;
17: pixel <= BLACK;

18: pixel <= BLACK; //3 ROW
19: pixel <= BLACK;
20: pixel <= RED;
21: pixel <= RED;
22: pixel <= RED;
23: pixel <= RED;
24: pixel <= RED;
25: pixel <= BLACK;
26: pixel <= BLACK;

27: pixel <= BLACK;//4 ROW
28: pixel <= BLACK;
29: pixel <= BLACK;
```

```
30: pixel <= RED;
31: pixel <= RED;
32: pixel <= RED;
33: pixel <= BLACK;
34: pixel <= BLACK;
35: pixel <= RED;

36: pixel <= BLACK;//5 ROW
37: pixel <= BLACK;
38: pixel <= BLACK;
39: pixel <= RED;
40: pixel <= RED;
41: pixel <= RED;
42: pixel <= BLACK;
43: pixel <= RED;
44: pixel <= BLACK;

45:pixel <= RED; //6 ROW
46:pixel <= RED;
47:pixel <= RED;
48:pixel <= RED;
49:pixel <= RED;
50:pixel <= RED;
51:pixel <= RED;
52:pixel <= BLACK;
53:pixel <= BLACK;

54: pixel <= BLACK;//7 ROW
55: pixel <= BLACK;
```

```
56: pixel <= BLACK;
57: pixel <= RED;
58: pixel <= RED;
59: pixel <= RED;
60: pixel <= BLACK;
61: pixel <= BLACK;
62: pixel <= BLACK;

63: pixel <= BLACK; //8 ROW
64: pixel <= BLACK;
65: pixel <= BLACK;
66: pixel <= RED;
67: pixel <= BLACK;
68: pixel <= RED;
69: pixel <= BLACK;
70: pixel <= BLACK;
71: pixel <= BLACK;

72: pixel <= BLACK; //9 ROW
73: pixel <= BLACK;
74: pixel <= RED;
75: pixel <= BLACK;
76: pixel <= BLACK;
77: pixel <= BLACK;
78: pixel <= RED;
79: pixel <= BLACK;
80: pixel <= BLACK;

81: pixel <= RED; //10 ROW
```

```
            82: pixel <= RED;
            83: pixel <= BLACK;
            84: pixel <= BLACK;
            85: pixel <= BLACK;
            86: pixel <= BLACK;
            87: pixel <= BLACK;
            88: pixel <= RED;
            89: pixel <= RED;
            default: pixel <= BLACK;
            endcase
    end
    else if (state == kneel_state) begin
            case(location)
            0: pixel <= BLACK; //ROW 1
            1: pixel <= BLACK;
            2: pixel <= RED;
            3: pixel <= RED;
            4: pixel <= RED;
            5: pixel <= RED;
            6: pixel <= RED;
            7: pixel <= BLACK;
            8: pixel <= BLACK;

            9: pixel <= BLACK; //ROW 2
            10: pixel <= BLACK;
            11: pixel <= RED;
            12: pixel <= RED;
            13: pixel <= RED;
            14: pixel <= RED;
```

```
15: pixel <= RED;
16: pixel <= BLACK;
17: pixel <= BLACK;

18: pixel <= BLACK; //3 ROW
19: pixel <= BLACK;
20: pixel <= RED;
21: pixel <= RED;
22: pixel <= RED;
23: pixel <= RED;
24: pixel <= RED;
25: pixel <= BLACK;
26: pixel <= BLACK;

27: pixel <= BLACK;//4 ROW
28: pixel <= BLACK;
29: pixel <= BLACK;
30: pixel <= RED;
31: pixel <= RED;
32: pixel <= RED;
33: pixel <= BLACK;
34: pixel <= BLACK;
35: pixel <= BLACK;

36: pixel <= BLACK;//5 ROW
37: pixel <= BLACK;
38: pixel <= BLACK;
39: pixel <= RED;
40: pixel <= RED;
```

```
41: pixel <= RED;
42: pixel <= BLACK;
43: pixel <= BLACK;
44: pixel <= BLACK;

45:pixel <= RED; //6 ROW
46:pixel <= RED;
47:pixel <= RED;
48:pixel <= RED;
49:pixel <= RED;
50:pixel <= RED;
51:pixel <= RED;
52:pixel <= RED;
53:pixel <= RED;

54: pixel <= BLACK;//7 ROW
55: pixel <= BLACK;
56: pixel <= BLACK;
57: pixel <= RED;
58: pixel <= RED;
59: pixel <= RED;
60: pixel <= BLACK;
61: pixel <= BLACK;
62: pixel <= BLACK;

63: pixel <= BLACK; //8 ROW
64: pixel <= RED;
65: pixel <= RED;
66: pixel <= RED;
```

```
                67: pixel <= BLACK;
                68: pixel <= RED;
                69: pixel <= RED;
                70: pixel <= RED;
                71: pixel <= BLACK;

                72: pixel <= RED; //9 ROW
                73: pixel <= RED;
                74: pixel <= BLACK;
                75: pixel <= BLACK;
                76: pixel <= BLACK;
                77: pixel <= BLACK;
                78: pixel <= BLACK;
                79: pixel <= RED;
                80: pixel <= RED;

                81: pixel <= BLACK; //10 ROW
                82: pixel <= BLACK;
                83: pixel <= BLACK;
                84: pixel <= BLACK;
                85: pixel <= BLACK;
                86: pixel <= BLACK;
                87: pixel <= BLACK;
                88: pixel <= BLACK;
                89: pixel <= BLACK;
                default: pixel <= BLACK;
                endcase
        end
end
```

```
        else pixel <= BLACK;
    end
endmodule
```

Marker_Convert.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:        21:53:20 12/04/2017
// Design Name:
// Module Name:      marker_convert
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

//This Module is to correlate the unknown marker to the specfic body part
module marker_convert(
    input clk,
```

```
input reset,
input [9:0] led_1_x, //head led
input [10:0] led_1_y,
input [23:0] led_1_rgb,
input [9:0] led_2_x, //body led
input [10:0] led_2_y,
input [23:0] led_2_rgb,
input [9:0] led_3_x, //left_arm_led
input [10:0] led_3_y,
input [23:0] led_3_rgb,
input [9:0] led_4_x, //right_arm_led
input [10:0] led_4_y,
input [23:0] led_4_rgb,
input [9:0] led_5_x, //left_leg_led, since in this case we just need one input
input [10:0] led_5_y,
input [23:0] led_5_rgb,
input [9:0] led_6_x, //right_leg_led, since in this case we just need one input
input [10:0] led_6_y,
input [23:0] led_6_rgb,

output reg [9:0] head_x,
output reg [10:0] head_y,
output reg [9:0] body_x,
output reg [10:0] body_y,
output reg [9:0] left_hand_x,
output reg [10:0] left_hand_y,
output reg [9:0] right_hand_x,
output reg [10:0] right_hand_y,
output reg [9:0] left_leg_x,
```

```
    output reg [10:0] left_leg_y,
    output reg [9:0] left_leg_x,
    output reg [10:0] left_leg_y,
    output reg front_back, // front = 1, back = 0
    output reg left_right // left = 1, right = 0
   );
    //temp storage
    reg [9:0] head_x_t;
    reg [10:0] head_y_t;
    reg [9:0] body_x_t;
   reg [10:0] body_y_t;
    reg [9:0] left_hand_x_t;
    reg [10:0] left_hand_y_t;
    reg [9:0] right_hand_x_t;
    reg [10:0] right_hand_y_t;
    reg [9:0] left_leg_x_t;
    reg [10:0] left_leg_y_t;
    reg [9:0] right_leg_x_t;
    reg [10:0] right_leg_y_t;


    always @ (posedge clk) begin
    // determine body orientation, front/back, left or
          if (led_1_rgb == ||led_2_rgb == || led_3_rgb ==||led_4_rgb == ||led_5_rgb == ||
led_6_rgb ) //yellow is detected
                front_back <= 0;
          else
                front_back <= 1;
    // determine the head
```

```
        head_x_t <= led_1_x;
        head_y_t <= led_1_y;
        if (head_y_t > led_2_y) begin
                head_x_t <= led_2_x;
                head_y_t <= led_2_y;
        end
        if (head_y_t > led_3_y) begin
                head_x_t <= led_3_x;
                head_y_t <= led_3_y;
        end
        if (head_y_t > led_4_y) begin
                head_x_t <= led_4_x;
                head_y_t <= led_4_y;
        end
        if (head_y_t > led_5_y) begin
                head_x_t <= led_5_x;
                head_y_t <= led_5_y;
        end
        if (head_y_t > led_6_y) begin
                head_x_t <= led_6_x;
                head_y_t <= led_6_y;
        end
        head_x <= head_x_t;
        head_y <= head_y_t;

// determine the left hand
        left_hand_x_t <= led_1_x;
        left_hand_y_t <= led_1_y;
```

```
        if (left_hand_x_t < led_2_x) begin
                left_hand_x_t <= led_2_x;
                left_hand_y_t <= led_2_y;
        end
        if (left_hand_x_t < led_3_x) begin
                left_hand_x_t <= led_3_x;
                left_hand_y_t <= led_3_y;
        end
        if (left_hand_x_t < led_4_x) begin
                left_hand_x_t <= led_4_x;
                left_hand_y_t <= led_4_y;
        end
        if (left_hand_x_t < led_5_x) begin
                left_hand_x_t <= led_5_x;
                left_hand_y_t <= led_5_y;
        end
        if (left_hand_x_t < led_6_x) begin
                left_hand_x_t <= led_6_x;
                left_hand_y_t <= led_6_y;
        end

        left_hand_x <= left_hand_x_t;
        left_hand_y <= left_hand_y_t;


// determine the right hand
        right_hand_x_t <= led_1_x;
        right_hand_y_t <= led_1_y;
        if (right_hand_x_t > led_2_x) begin
```

```
            right_hand_x_t <= led_2_x;
            right_hand_y_t <= led_2_y;
        end
        if (right_hand_x_t > led_3_x) begin
            right_hand_x_t <= led_3_x;
            right_hand_y_t <= led_3_y;
        end
        if (right_hand_x_t > led_4_x) begin
            right_hand_x_t <= led_4_x;
            right_hand_y_t <= led_4_y;
        end
        if (right_hand_x_t > led_5_x) begin
            right_hand_x_t <= led_5_x;
            right_hand_y_t <= led_5_y;
        end
        if (right_hand_x_t > led_6_x) begin
            right_hand_x_t <= led_6_x;
            right_hand_y_t <= led_6_y;
        end

        right_hand_x <= right_hand_x_t;
        right_hand_y <= right_hand_y_t;


   // determine the leg
        if ((led_1_y > right_hand_y) && (led_1_y > left_hand_y)&&(led_1_x > head_x))
begin // left leg led below hand and on the left
```

```
                    // of center body
                left_leg_x_t <= led_1_x;
                left_leg_y_t <= led_1_y;
            end
            else if ((led_1_y > right_hand_y) && (led_1_y >left_hand_y)&&(led_1_x <
head_x)) begin // right leg led below hand and on the left


                    // of center body
                right_leg_x_t <= led_1_x;
                right_leg_y_t <= led_1_y;
            end
            if ((led_2_y > right_hand_y) && (led_2_y > left_hand_y)&&(led_2_x > head_x))
begin // left leg led below hand and on the left


                    // of center body
                left_leg_x_t <= led_2_x;
                left_leg_y_t <= led_2_y;
            end
            else if ((led_2_y > right_hand_y) && (led_2_y > left_hand_y)&&(led_2_x <
head_x)) begin // right leg led below hand and on the left


                    // of center body
                right_leg_x_t <= led_2_x;
                right_leg_y_t <= led_2_y;
```

```
        end

        if ((led_3_y > right_hand_y) && (led_3_y > left_hand_y)&&(led_3_x > head_x))
begin // left leg led below hand and on the left


                // of center body
            left_leg_x_t <= led_3_x;
            left_leg_y_t <= led_3_y;
        end
        else if ((led_3_y > right_hand_y) && (led_3_y > left_hand_y)&&(led_3_x <
head_x)) begin // right leg led below hand and on the left


                // of center body
            right_leg_x_t <= led_3_x;
            right_leg_y_t <= led_3_y;
        end

        if ((led_4_y > right_hand_y) && (led_4_y > left_hand_y)&&(led_4_x > head_x))
begin // left leg led below hand and on the left


                // of center body
            left_leg_x_t <= led_4_x;
            left_leg_y_t <= led_4_y;
        end
        else if ((led_4_y > right_hand_y) && (led_4_y > left_hand_y)&&(led_4_x <
head_x)) begin // right leg led below hand and on the left
```

```
                    // of center body
            right_leg_x_t <= led_4_x;
            right_leg_y_t <= led_4_y;
        end

        if ((led_5_y > right_hand_y) && (led_5_y > left_hand_y)&&(led_5_x > head_x))
begin // left leg led below hand and on the left


                    // of center body
            left_leg_x_t <= led_1_x;
            left_leg_y_t <= led_1_y;
        end
        else if ((led_5_y >right_hand_y) && (led_5_y > left_hand_y)&&(led_5_x <
head_x)) begin // right leg led below hand and on the left


                    // of center body
            right_leg_x_t <= led_1_x;
            right_leg_y_t <= led_1_y;
        end

        if ((led_6_y > right_hand_y) && (led_6_y > left_hand_y)&&(led_6_x > head_x))
begin // left leg led below hand and on the left


                    // of center body
```

```
                left_leg_x_t <= led_6_x;
                left_leg_y_t <= led_6_y;
        end
        else if ((led_6_y > right_hand_y) && (led_6_y > left_hand_y)&&(led_6_x <
head_x)) begin // right leg led below hand and on the left


                        // of center body
                right_leg_x_t <= led_6_x;
                right_leg_y_t <= led_6_y;
        end

        left_leg_x <= left_leg_x_t;
        left_leg_y <= left_leg_y_t;
        right_leg_x <= right_leg_x_t;
        right_leg_y <= right_leg_y_t;
    end
endmodule
```

display.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:         19:28:35 11/20/2017
// Design Name:
// Module Name:      puck_conversion_two
// Project Name:
```

```
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments: ools /mit/6.1/tools.tcsh

//
//////////////////////////////////////////////////////////////////////////////////
module Display(
    input clk,
    input [10:0] hcount, // syncronize the conversion module with frame rate of VGA
    input [9:0] vcount,
    input [3:0] motion,
    input up,
    input down,
    input left,
    input right,
    input jump,
    input left_wave,
    input right_wave,
    input kneel,
    input reset,
    input left_mouse,
    input right_mouse,
    input up_mouse,
```

```
input down_mouse,
output [23:0] p_out,
output reg [7:0] keyboard,
output reg [3:0] mouse,
output reg [7:0] led
      );
//initilize
parameter BLACK = 24'h00_00_00;
parameter RED = 24'hff_00_00;
parameter TAN = 24'hde_b8_87;
parameter BROWN = 24'h99_66_33;
parameter CLEAR = 24'h00_00_00;
// parameter SCALE = 4; //power of two scaling factor
parameter HEIGHT = 10;
parameter WIDTH = 9;

// action state parameters
parameter up_state = 4'b1001;
parameter down_state = 4'b0001;
parameter left_state = 4'b0010;
parameter right_state = 4'b0011;
parameter jump_state = 4'b0100;
parameter left_wave_state = 4'b0101;
parameter right_wave_state = 4'b0110;
parameter halt_state = 4'b0111;
parameter kneel_state = 4'b1000;

reg [23:0] pixel;
reg [7:0] location;
```

```
    reg [10:0] current_x;
    reg [9:0] current_y;
    reg [1:0] current_scale;
    reg jump_loop; //this register is to make sure that the jump loop only happens once
for every jump input
    reg [3:0] state;
    // assign scale = current_scale;
    assign p_out = pixel;
    reg up_flag;
    reg [4:0] jump_counter;

    always @ (posedge clk) begin
        // testing purpose
        /*
            4'b0000;
            4'b0001;
            4'b0010;
            4'b0011;
            4'b0100;
            4'b0101;
            4'b0110;
            4'b0111;
        */
        //deermine state and keyboard press output

        if (up || motion == up_state) begin
            state <= 4'b1001;
            keyboard <= 8'b11111110;
            led <= 8'b00000001; // led marker to indicate the action detected
```

```
        end
        else if (down || motion == down_state) begin
                state <= 4'b0001;
                keyboard <= 8'b11111101;
                led <= 8'b00000010;
        end
        else if (left || motion == left_state) begin
                state <=4'b0010;
                keyboard <= 8'b11111011;
        ools /mit/6.1/tools.tcsh
led <= 8'b00000100;
        end
        else if (right || motion == right_state) begin
                state <=4'b0011;
                keyboard <= 8'b11110111;
                led <= 8'b00001000;
        end
        else if (jump|| motion == jump_state) begin
                state <=4'b0100;
                keyboard <= 8'b11101111;
                led <= 8'b00010000;
        end
        else if (left_wave|| motion == left_wave_state) begin
                state <=4'b0101;
                // keyboard <= 8'b11011111;
                mouse <= 4'b0001; //we will use left wave for moving mouse right
                led <= 8'b00100000;
        end
        else if (right_wave|| motion == right_wave_state) begin
```

```
        state <=4'b0110;
        // keyboard <= 8'b10111111;
        mouse <= 4'b0010; // we will use right wave for moving mouse left
        led <= 8'b01000000;
end
else if (kneel|| motion == kneel_state) begin
        state <= 4'b1000;
        // keyboard <= 8'b01111111;
        mouse <= 4'b1000; // move mouse down
        led <= 8'b10000000;
end
else if (motion == halt_state) begin// halt state
        state <=4'b0111;
        keyboard <= 8'b11111111;
        mouse <= 4'b0000;
        led <= 8'b00000000;
end
else begin
        state <=4'b0111; ools /mit/6.1/tools.tcsh

        keyboard <= 8'b11111111;
        mouse <= 4'b0000;
        led <= 8'b00000000;
end

// gun/mouse movement
/*
if (left_mouse)
        mouse <= 4'b0001;
```

```
else if (right_mouse)
        mouse <= 4'b0010;
else if (up_mouse)
        mouse <= 4'b0100;
else if (down_mouse)
        mouse <= 4'b1000;
else
        mouse <= 4'b0000;
*/
//VGA display section

if (hcount == 0&& vcount == 0) begin

        if (reset) begin
                current_x <= 10'd512;
                current_y <= 9'd384;
                current_scale <= 2'd3;
                state <= halt_state;
        end

        else if (state == up_state) begin
                if (current_y < 10)
                        current_y <= 0;
                else
                        current_y <= current_y - 10;
                state <= halt_state;
        end

        else if (state == down_state) begin
```

```
            if (current_y > 680)
                current_y <= 681;
            else
                current_y <= current_y + 10;
            state <= halt_state;
        end

        else if (state == left_state) begin
            if (current_x < 10)
                current_x <= 0;
            else
                current_x <= current_x - 10;
            state <= halt_state;
        end

        else if (state == right_state) begin // because of the scaling factor, we
want to show the whoe figure, so that we will reserve more area
                                                    // on right
            if (current_x >952) begin
                current_x <= 953;
                state <= halt_state;
            end
            else begin
                current_x <= current_x + 10;
                state <= halt_state;
            end
        end
```

```
else if (state == jump_state) begin // for the jump out put, we will change on the scaling factor
                if (jump_counter < 5'd18) begin
                        if (up_flag) begin
                                jump_counter <= jump_counter + 1;
                                if (jump_counter == 5'd3)
                                        current_scale <= current_scale - 1;

                                if (jump_counter == 5'd6)
                                        current_scale <= current_scale - 1;
                                if (jump_counter == 5'd9)
                                        current_scale <= current_scale - 1;
                                if (current_scale == 2'd3)
                                        up_flag <= 0;
                        end
                        else if (~up_flag) begin
                                jump_counter <= jump_counter + 1;
                                if (jump_counter == 5'd12)
                                        current_scale <= current_scale + 1;

                                if (jump_counter == 5'd15)
                                        current_scale <= current_scale + 1;
                                if (jump_counter == 5'd17)
                                        current_scale <= current_scale + 1;
                        end
                end
                else if (jump_counter >= 5'd18) begin
                        jump_counter <= 0;
                        current_scale <= 2'd3;
```

```
                                    state <= halt_state; // one loop has been finished
                                end
                    end

                    // else state <= halt_state;
            end
            // puck display section
            if (((hcount-1) >= current_x && (hcount-1) <
(current_x+(WIDTH<<current_scale))) && ((vcount + 1) >= current_y && (vcount+ 1) <
(current_y+(HEIGHT<<current_scale)))) begin

            //test
            // if ((hcount >= 100 && hcount < 200) && (vcount >= 100 && vcount < 200))
begin
                    location <= ((hcount-current_x)>>current_scale) + (((vcount -
current_y))>>current_scale)*WIDTH;
                    //test
                    //location <= hcount>>1;
                    if (state == halt_state || state == up_state || state == left_state || state ==
right_state || state == down_state || state == jump_state ) begin
                            case(location)
                            0: pixel <= BLACK; //ROW 1
                            1: pixel <= BLACK;
                            2: pixel <= RED;
                            3: pixel <= RED;
                            4: pixel <= RED;
                            5: pixel <= RED;
                            6: pixel <= RED;
                            7: pixel <= BLACK;
```

```
8: pixel <= BLACK;

9: pixel <= BLACK; //ROW 2
10: pixel <= BLACK;
11: pixel <= RED;
12: pixel <= RED;
13: pixel <= RED;
14: pixel <= RED;
15: pixel <= RED;
16: pixel <= BLACK;
17: pixel <= BLACK;

18: pixel <= BLACK; //3 ROW
19: pixel <= BLACK;
20: pixel <= RED;
21: pixel <= RED;
22: pixel <= RED;
23: pixel <= RED;
24: pixel <= RED;
25: pixel <= BLACK;
26: pixel <= BLACK;

27: pixel <= BLACK;//4 ROW
28: pixel <= BLACK;
29: pixel <= BLACK;
30: pixel <= RED;
31: pixel <= RED;
32: pixel <= RED;
33: pixel <= BLACK;
```

```
34: pixel <= BLACK;
35: pixel <= BLACK;

36: pixel <= BLACK;//5 ROW
37: pixel <= BLACK;
38: pixel <= BLACK;
39: pixel <= RED;
40: pixel <= RED;
41: pixel <= RED;
42: pixel <= BLACK;
43: pixel <= BLACK;
44: pixel <= BLACK;

45:pixel <= RED; //6 ROW
46:pixel <= RED;
47:pixel <= RED;
48:pixel <= RED;
49:pixel <= RED;
50:pixel <= RED;
51:pixel <= RED;
52:pixel <= RED;
53:pixel <= RED;

54: pixel <= BLACK;//7 ROW
55: pixel <= BLACK;
56: pixel <= BLACK;
57: pixel <= RED;
58: pixel <= RED;
59: pixel <= RED;
```

```
60: pixel <= BLACK;
61: pixel <= BLACK;
62: pixel <= BLACK;

63: pixel <= BLACK; //8 ROW
64: pixel <= BLACK;
65: pixel <= BLACK;
66: pixel <= RED;
67: pixel <= BLACK;
68: pixel <= RED;
69: pixel <= BLACK;
70: pixel <= BLACK;
71: pixel <= BLACK;

72: pixel <= BLACK; //9 ROW
73: pixel <= BLACK;
74: pixel <= RED;
75: pixel <= BLACK;
76: pixel <= BLACK;
77: pixel <= BLACK;
78: pixel <= RED;
79: pixel <= BLACK;
80: pixel <= BLACK;


81: pixel <= RED; //10 ROW
82: pixel <= RED;
83: pixel <= BLACK;
84: pixel <= BLACK;
```

```
            85: pixel <= BLACK;
            86: pixel <= BLACK;
            87: pixel <= BLACK;
            88: pixel <= RED;
            89: pixel <= RED;

            default: pixel <= BLACK;
            endcase
    end
    else if (state == left_wave_state) begin
            case(location)
            0: pixel <= BLACK; //ROW 1 head
            1: pixel <= BLACK;
            2: pixel <= RED;
            3: pixel <= RED;
            4: pixel <= RED;
            5: pixel <= RED;
            6: pixel <= RED;
            7: pixel <= BLACK;
            8: pixel <= BLACK;

            9: pixel <= BLACK; //ROW 2
            10: pixel <= BLACK;
            11: pixel <= RED;
            12: pixel <= RED;
            13: pixel <= RED;
            14: pixel <= RED;
            15: pixel <= RED;
            16: pixel <= BLACK;
```

```
17: pixel <= BLACK;

18: pixel <= BLACK; //3 ROW
19: pixel <= BLACK;
20: pixel <= RED;
21: pixel <= RED;
22: pixel <= RED;

23: pixel <= RED;
24: pixel <= RED;
25: pixel <= BLACK;
26: pixel <= BLACK;

27: pixel <= RED;//4 ROW neck
28: pixel <= BLACK;
29: pixel <= BLACK;
30: pixel <= RED;
31: pixel <= RED;
32: pixel <= RED;
33: pixel <= BLACK;
34: pixel <= BLACK;
35: pixel <= BLACK;

36: pixel <= BLACK;//5 ROW
37: pixel <= RED;
38: pixel <= BLACK;
39: pixel <= RED;
40: pixel <= RED;
41: pixel <= RED;
```

```
42: pixel <= BLACK;
43: pixel <= BLACK;
44: pixel <= BLACK;

45:pixel <= BLACK; //6 ROW hand
46:pixel <= BLACK;
47:pixel <= RED;
48:pixel <= RED;
49:pixel <= RED;
50:pixel <= RED;
51:pixel <= RED;
52:pixel <= RED;
53:pixel <= RED;

54: pixel <= BLACK;//7 ROW     WAIST
55: pixel <= BLACK;
56: pixel <= BLACK;
57: pixel <= RED;
58: pixel <= RED;
59: pixel <= RED;
60: pixel <= BLACK;
61: pixel <= BLACK;
62: pixel <= BLACK;

63: pixel <= BLACK; //8 ROW LEG
64: pixel <= BLACK;
65: pixel <= BLACK;
66: pixel <= RED;
67: pixel <= BLACK;
```

```
        68: pixel <= RED;
        69: pixel <= BLACK;
        70: pixel <= BLACK;
        71: pixel <= BLACK;

        72: pixel <= BLACK; //9 ROW
        73: pixel <= BLACK;
        74: pixel <= RED;
        75: pixel <= BLACK;
        76: pixel <= BLACK;
        77: pixel <= BLACK;
        78: pixel <= RED;
        79: pixel <= BLACK;
        80: pixel <= BLACK;

        81: pixel <= RED; //10 ROW
        82: pixel <= RED;
        83: pixel <= BLACK;
        84: pixel <= BLACK;
        85: pixel <= BLACK;
        86: pixel <= BLACK;
        87: pixel <= BLACK;
        88: pixel <= RED;
        89: pixel <= RED;
        default: pixel <= BLACK;
        endcase
end
else if (state == right_wave_state) begin
case(location)
```

```
0: pixel <= BLACK; //ROW 1
1: pixel <= BLACK;
2: pixel <= RED;
3: pixel <= RED;
4: pixel <= RED;
5: pixel <= RED;
6: pixel <= RED;
7: pixel <= BLACK;
8: pixel <= BLACK;

9: pixel <= BLACK; //ROW 2
10: pixel <= BLACK;
11: pixel <= RED;
12: pixel <= RED;
13: pixel <= RED;
14: pixel <= RED;
15: pixel <= RED;
16: pixel <= BLACK;
17: pixel <= BLACK;

18: pixel <= BLACK; //3 ROW
19: pixel <= BLACK;
20: pixel <= RED;
21: pixel <= RED;
22: pixel <= RED;
23: pixel <= RED;
24: pixel <= RED;
25: pixel <= BLACK;
26: pixel <= BLACK;
```

```
27: pixel <= BLACK;//4 ROW
28: pixel <= BLACK;
29: pixel <= BLACK;
30: pixel <= RED;
31: pixel <= RED;
32: pixel <= RED;
33: pixel <= BLACK;
34: pixel <= BLACK;
35: pixel <= RED;

36: pixel <= BLACK;//5 ROW
37: pixel <= BLACK;
38: pixel <= BLACK;
39: pixel <= RED;
40: pixel <= RED;
41: pixel <= RED;
42: pixel <= BLACK;
43: pixel <= RED;
44: pixel <= BLACK;

45:pixel <= RED; //6 ROW
46:pixel <= RED;
47:pixel <= RED;
48:pixel <= RED;
49:pixel <= RED;
50:pixel <= RED;
51:pixel <= RED;
52:pixel <= BLACK;
```

```
53:pixel <= BLACK;

54: pixel <= BLACK;//7 ROW
55: pixel <= BLACK;
56: pixel <= BLACK;
57: pixel <= RED;
58: pixel <= RED;
59: pixel <= RED;
60: pixel <= BLACK;
61: pixel <= BLACK;
62: pixel <= BLACK;

63: pixel <= BLACK; //8 ROW
64: pixel <= BLACK;
65: pixel <= BLACK;
66: pixel <= RED;
67: pixel <= BLACK;
68: pixel <= RED;
69: pixel <= BLACK;
70: pixel <= BLACK;
71: pixel <= BLACK;

72: pixel <= BLACK; //9 ROW
73: pixel <= BLACK;
74: pixel <= RED;
75: pixel <= BLACK;
76: pixel <= BLACK;
77: pixel <= BLACK;
78: pixel <= RED;
```

```
            79: pixel <= BLACK;
            80: pixel <= BLACK;

            81: pixel <= RED; //10 ROW
            82: pixel <= RED;
            83: pixel <= BLACK;
            84: pixel <= BLACK;
            85: pixel <= BLACK;
            86: pixel <= BLACK;
            87: pixel <= BLACK;
            88: pixel <= RED;
            89: pixel <= RED;
            default: pixel <= BLACK;
            endcase
    end
    else if (state == kneel_state) begin
            case(location)
            0: pixel <= BLACK; //ROW 1
            1: pixel <= BLACK;
            2: pixel <= RED;
            3: pixel <= RED;
            4: pixel <= RED;
            5: pixel <= RED;
            6: pixel <= RED;
            7: pixel <= BLACK;
            8: pixel <= BLACK;

            9: pixel <= BLACK; //ROW 2
            10: pixel <= BLACK;
```

```
11: pixel <= RED;
12: pixel <= RED;
13: pixel <= RED;
14: pixel <= RED;
15: pixel <= RED;
16: pixel <= BLACK;
17: pixel <= BLACK;

18: pixel <= BLACK; //3 ROW
19: pixel <= BLACK;
20: pixel <= RED;
21: pixel <= RED;
22: pixel <= RED;
23: pixel <= RED;
24: pixel <= RED;
25: pixel <= BLACK;
26: pixel <= BLACK;

27: pixel <= BLACK;//4 ROW
28: pixel <= BLACK;
29: pixel <= BLACK;
30: pixel <= RED;
31: pixel <= RED;
32: pixel <= RED;
33: pixel <= BLACK;
34: pixel <= BLACK;
35: pixel <= BLACK;

36: pixel <= BLACK;//5 ROW
```

```
37: pixel <= BLACK;
38: pixel <= BLACK;
39: pixel <= RED;
40: pixel <= RED;
41: pixel <= RED;
42: pixel <= BLACK;
43: pixel <= BLACK;
44: pixel <= BLACK;

45:pixel <= RED; //6 ROW
46:pixel <= RED;
47:pixel <= RED;
48:pixel <= RED;
49:pixel <= RED;
50:pixel <= RED;
51:pixel <= RED;
52:pixel <= RED;
53:pixel <= RED;

54: pixel <= BLACK;//7 ROW
55: pixel <= BLACK;
56: pixel <= BLACK;
57: pixel <= RED;
58: pixel <= RED;
59: pixel <= RED;
60: pixel <= BLACK;
61: pixel <= BLACK;
62: pixel <= BLACK;
```

```
63: pixel <= BLACK; //8 ROW
64: pixel <= RED;
65: pixel <= RED;
66: pixel <= RED;
67: pixel <= BLACK;
68: pixel <= RED;
69: pixel <= RED;
70: pixel <= RED;
71: pixel <= BLACK;

72: pixel <= RED; //9 ROW
73: pixel <= RED;
74: pixel <= BLACK;
75: pixel <= BLACK;
76: pixel <= BLACK;
77: pixel <= BLACK;
78: pixel <= BLACK;
79: pixel <= RED;
80: pixel <= RED;

81: pixel <= BLACK; //10 ROW
82: pixel <= BLACK;
83: pixel <= BLACK;
84: pixel <= BLACK;
85: pixel <= BLACK;
86: pixel <= BLACK;
87: pixel <= BLACK;
88: pixel <= BLACK;
89: pixel <= BLACK;
```

```
                        default: pixel <= BLACK;
                        endcase
                end
        end
        else pixel <= BLACK;
   end
endmodule
```

zbt_6111_sample.v

```
//
// File:   zbt_6111_sample.v
// Date:   26-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Sample code for the MIT 6.111 labkit demonstrating use of the ZBT
// memories for video display.  Video input from the NTSC digitizer is
// displayed within an XGA 1024x768 window.  One ZBT memory (ram0) is used
// as the video frame buffer, with 8 bits used per pixel (black & white).
//
// Since the ZBT is read once for every four pixels, this frees up time for
// data to be stored to the ZBT during other pixel times.  The NTSC decoder
// runs at 27 MHz, whereas the XGA runs at 65 MHz, so we synchronize
// signals between the two (see ntsc2zbt.v) and let the NTSC data be
// stored to ZBT memory whenever it is available, during cycles when
// pixel reads are not being performed.
//
// We use a very simple ZBT interface, which does not involve any clock
// generation or hiding of the pipelining.  See zbt_6111.v for more info.
//
```

```
// switch[7] selects between display of NTSC video and test bars
// switch[6] is used for testing the NTSC decoder
// switch[1] selects between test bar periods; these are stored to ZBT
//         during blanking periods
// switch[0] selects vertical test bars (hardwired; not stored in ZBT)
//
//
// Bug fix: Jonathan P. Mailoa <jpmailoa@mit.edu>
// Date   : 11-May-09
//
// Use ramclock module to deskew clocks;  GPH
// To change display from 1024*787 to 800*600, use clock_40mhz and change
// accordingly. Verilog ntsc2zbt.v will also need changes to change resolution.
//
// Date   : 10-Nov-11

//////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////
//
// CHANGES FOR BOARD REVISION 004
```

```
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.
// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
//       "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
//       output of the FPGA, and "in" is an input.
//
// CHANGES FOR BOARD REVISION 003
//
// 1) Combined flash chip enables into a single signal, flash_ce_b.
//
// CHANGES FOR BOARD REVISION 002
//
// 1) Added SRAM clock feedback path input and output
// 2) Renamed "mousedata" to "mouse_data"
// 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
//       the data bus, and the byte write enables have been combined into the
//       4-bit ram#_bwe_b bus.
// 4) Removed the "systemace_clock" net, since the SystemACE clock is now
//       hardwired on the PCB to the oscillator.
//
//////////////////////////////////////////////////////////////////////////////
//
// Complete change history (including bug fixes)
//
// 2011-Nov-10: Changed resolution to 1024 * 768.
//                  Added back ramclok to deskew RAM clock
//
```

```
// 2009-May-11: Fixed memory management bug by 8 clock cycle forecast.
//              Changed resolution to  800 * 600.
//              Reduced clock speed to 40MHz.
//              Disconnected zbt_6111's ram_clk signal.
//              Added ramclock to control RAM.
//              Added notes about ram1 default values.
//              Commented out clock_feedback_out assignment.
//              Removed delayN modules because ZBT's latency has no more effect.
//
// 2005-Sep-09: Added missing default assignments to "ac97_sdata_out",
//              "disp_data_out", "analyzer[2-3]_clock" and
//              "analyzer[2-3]_data".
//
// 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
//              actually populated on the boards. (The boards support up to
//              256Mb devices, with 25 address lines.)
//
// 2004-Oct-31: Adapted to new revision 004 board.
//
// 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
//              value. (Previous versions of this file declared this port to
//              be an input.)
//
// 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
//              actually populated on the boards. (The boards support up to
//              72Mb devices, with 21 address lines.)
//
// 2004-Apr-29: Change history started
//
```

```
//////////////////////////////////////////////////////////////////////////////

module zbt_6111_sample(beep, audio_reset_b,
                       ac97_sdata_out, ac97_sdata_in, ac97_synch,
                ac97_bit_clock,

                vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
                vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
                vga_out_vsync,

                tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
                tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
                tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,

                tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
                tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
                tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
                tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,

                ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
                ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,

                ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
                ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,

                clock_feedback_out, clock_feedback_in,

                flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
                flash_reset_b, flash_sts, flash_byte_b,
```

rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

mouse_clock, mouse_data, keyboard_clock, keyboard_data,

clock_27mhz, clock1, clock2,

disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
disp_reset_b, disp_data_in,

button0, button1, button2, button3, button_enter, button_right,
button_left, button_down, button_up,

switch,

led,

user1, user2, user3, user4,

daughtercard,

systemace_data, systemace_address, systemace_ce_b,
systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,

analyzer1_data, analyzer1_clock,
analyzer2_data, analyzer2_clock,
analyzer3_data, analyzer3_clock,
analyzer4_data, analyzer4_clock);

```
output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
input  ac97_bit_clock, ac97_sdata_in;

output [7:0] vga_out_red, vga_out_green, vga_out_blue;
output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
       vga_out_hsync, vga_out_vsync;

output [9:0] tv_out_ycrcb;
output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
       tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
       tv_out_subcar_reset;

input  [19:0] tv_in_ycrcb;
input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
       tv_in_hff, tv_in_aff;
output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
       tv_in_reset_b, tv_in_clock;
inout  tv_in_i2c_data;

inout  [35:0] ram0_data;
output [18:0] ram0_address;
output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
output [3:0] ram0_bwe_b;

inout  [35:0] ram1_data;
output [18:0] ram1_address;
output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
output [3:0] ram1_bwe_b;
```

```
input  clock_feedback_in;
output clock_feedback_out;

inout  [15:0] flash_data;
output [23:0] flash_address;
output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
input  flash_sts;

output rs232_txd, rs232_rts;
input  rs232_rxd, rs232_cts;

input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

input  clock_27mhz, clock1, clock2;

output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
input  disp_data_in;
output  disp_data_out;

input  button0, button1, button2, button3, button_enter, button_right,
       button_left, button_down, button_up;
input  [7:0] switch;
output [7:0] led;

inout [31:0] user1, user2, user3, user4;

inout [43:0] daughtercard;

inout  [15:0] systemace_data;
```

```
    output [6:0]  systemace_address;
    output systemace_ce_b, systemace_we_b, systemace_oe_b;
    input  systemace_irq, systemace_mpbrdy;

    output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
           analyzer4_data;
    output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;

    //////////////////////////////////////////////////////////////////////
    //
    // I/O Assignments
    //
    //////////////////////////////////////////////////////////////////////

    // Audio Input and Output
    assign beep= 1'b0;
    assign audio_reset_b = 1'b0;
    assign ac97_synch = 1'b0;
    assign ac97_sdata_out = 1'b0;
/*
*/
    // ac97_sdata_in is an input

    // Video Output
    assign tv_out_ycrcb = 10'h0;
    assign tv_out_reset_b = 1'b0;
    assign tv_out_clock = 1'b0;
    assign tv_out_i2c_clock = 1'b0;
    assign tv_out_i2c_data = 1'b0;
```

```
   assign tv_out_pal_ntsc = 1'b0;
   assign tv_out_hsync_b = 1'b1;
   assign tv_out_vsync_b = 1'b1;
   assign tv_out_blank_b = 1'b1;
   assign tv_out_subcar_reset = 1'b0;

   // Video Input
   //assign tv_in_i2c_clock = 1'b0;
   assign tv_in_fifo_read = 1'b1;
   assign tv_in_fifo_clock = 1'b0;
   assign tv_in_iso = 1'b1;
   //assign tv_in_reset_b = 1'b0;
   assign tv_in_clock = clock_27mhz;//1'b0;
   //assign tv_in_i2c_data = 1'bZ;
   // tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
   // tv_in_aef, tv_in_hff, and tv_in_aff are inputs

   // SRAMs

/* change lines below to enable ZBT RAM bank0 */

/*
   assign ram0_data = 36'hZ;
   assign ram0_address = 19'h0;
   assign ram0_clk = 1'b0;
   assign ram0_we_b = 1'b1;
   assign ram0_cen_b = 1'b0;       // clock enable
*/
```

```
/* enable RAM pins */

   assign ram0_ce_b = 1'b0;
   assign ram0_oe_b = 1'b0;
   assign ram0_adv_ld = 1'b0;
   assign ram0_bwe_b = 4'h0;

/**********/

   assign ram1_data = 36'hZ;
   assign ram1_address = 19'h0;
   assign ram1_adv_ld = 1'b0;
   assign ram1_clk = 1'b0;

   //These values has to be set to 0 like ram0 if ram1 is used.
   assign ram1_cen_b = 1'b1;
   assign ram1_ce_b = 1'b1;
   assign ram1_oe_b = 1'b1;
   assign ram1_we_b = 1'b1;
   assign ram1_bwe_b = 4'hF;

   // clock_feedback_out will be assigned by ramclock
   // assign clock_feedback_out = 1'b0;  //2011-Nov-10
   // clock_feedback_in is an input

   // Flash ROM
   assign flash_data = 16'hZ;
   assign flash_address = 24'h0;
   assign flash_ce_b = 1'b1;
```

```
  assign flash_oe_b = 1'b1;
  assign flash_we_b = 1'b1;
  assign flash_reset_b = 1'b0;
  assign flash_byte_b = 1'b1;
  // flash_sts is an input

  // RS-232 Interface
  assign rs232_txd = 1'b1;
  assign rs232_rts = 1'b1;
  // rs232_rxd and rs232_cts are inputs

  // PS/2 Ports
  // mouse_clock, mouse_data, keyboard_clock, and keyboard_data are inputs

  // LED Displays
/*
  assign disp_blank = 1'b1;
  assign disp_clock = 1'b0;
  assign disp_rs = 1'b0;
  assign disp_ce_b = 1'b1;
  assign disp_reset_b = 1'b0;
  assign disp_data_out = 1'b0;
*/
  // disp_data_in is an input

  // Buttons, Switches, and Individual LEDs
  //lab3 assign led = 8'hFF;
  // button0, button1, button2, button3, button_enter, button_right,
  // button_left, button_down, button_up, and switches are inputs
```

```
// User I/Os
assign user1 = 32'hZ;
assign user2 = 32'hZ;
assign user3 = 32'hZ;
assign user4 = 32'hZ;

// Daughtercard Connectors
assign daughtercard = 44'hZ;

// SystemACE Microprocessor Port
assign systemace_data = 16'hZ;
assign systemace_address = 7'h0;
assign systemace_ce_b = 1'b1;
assign systemace_we_b = 1'b1;
assign systemace_oe_b = 1'b1;
// systemace_irq and systemace_mpbrdy are inputs

// Logic Analyzer
assign analyzer1_data = 16'h0;
assign analyzer1_clock = 1'b1;
assign analyzer2_data = 16'h0;
assign analyzer2_clock = 1'b1;
assign analyzer3_data = 16'h0;
assign analyzer3_clock = 1'b1;
assign analyzer4_data = 16'h0;
assign analyzer4_clock = 1'b1;

//////////////////////////////////////////////////////////////////////////
```

```
   // Demonstration of ZBT RAM as video memory

   // use FPGA's digital clock manager to produce a
   // 65MHz clock (actually 64.8MHz)
   wire clock_65mhz_unbuf,clock_65mhz;
   DCM vclk1(.CLKIN(clock_27mhz),.CLKFX(clock_65mhz_unbuf));
   // synthesis attribute CLKFX_DIVIDE of vclk1 is 10
   // synthesis attribute CLKFX_MULTIPLY of vclk1 is 24
   // synthesis attribute CLK_FEEDBACK of vclk1 is NONE
   // synthesis attribute CLKIN_PERIOD of vclk1 is 37
   BUFG vclk2(.O(clock_65mhz),.I(clock_65mhz_unbuf));

//   wire clk = clock_65mhz;  // gph 2011-Nov-10

/*   //////////////////////////////////////////////////////////////////////////
   // Demonstration of ZBT RAM as video memory

   // use FPGA's digital clock manager to produce a
   // 40MHz clock (actually 40.5MHz)
   wire clock_40mhz_unbuf,clock_40mhz;
   DCM vclk1(.CLKIN(clock_27mhz),.CLKFX(clock_40mhz_unbuf));
   // synthesis attribute CLKFX_DIVIDE of vclk1 is 2
   // synthesis attribute CLKFX_MULTIPLY of vclk1 is 3
   // synthesis attribute CLK_FEEDBACK of vclk1 is NONE
   // synthesis attribute CLKIN_PERIOD of vclk1 is 37
   BUFG vclk2(.O(clock_40mhz),.I(clock_40mhz_unbuf));

   wire clk = clock_40mhz;
*/
```

```
        wire locked;
        //assign clock_feedback_out = 0; // gph 2011-Nov-10

ramclock rc(.ref_clock(clock_65mhz), .fpga_clock(clk),
                .ram0_clock(ram0_clk),
                //.ram1_clock(ram1_clk),   //uncomment if ram1 is used
                .clock_feedback_in(clock_feedback_in),
                .clock_feedback_out(clock_feedback_out), .locked(locked));




// power-on reset generation
wire power_on_reset;      // remain high for first 16 clocks
SRL16 reset_sr (.D(1'b0), .CLK(clk), .Q(power_on_reset),
        .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
defparam reset_sr.INIT = 16'hFFFF;

// ENTER button is user reset
wire reset,user_reset, left, right, up, down;
debounce db1(power_on_reset, clk, ~button_enter, user_reset);
 debounce db2(power_on_reset, clk, ~button_left, left);
 debounce db3(power_on_reset, clk, ~button_right, right);
debounce db4(power_on_reset, clk, ~button_up, up);
 debounce db5(power_on_reset, clk, ~button_down, down);

 assign reset = user_reset | power_on_reset;

// display module for debugging
```

```
reg [63:0] dispdata;
display_16hex hexdisp1(reset, clk, dispdata,
       disp_blank, disp_clock, disp_rs, disp_ce_b,
       disp_reset_b, disp_data_out);

// generate basic XVGA video signals
wire [10:0] hcount;
wire [9:0]  vcount;
wire hsync,vsync,blank;
xvga xvga1(clk,hcount,vcount,hsync,vsync,blank);

// wire up to ZBT ram

wire [35:0] vram_write_data;
wire [35:0] vram_read_data;
wire [18:0] vram_addr;
wire        vram_we;

wire ram0_clk_not_used;
zbt_6111 zbt1(clk, 1'b1, vram_we, vram_addr,
       vram_write_data, vram_read_data,
       ram0_clk_not_used,   //to get good timing, don't connect ram_clk to zbt_6111
       ram0_we_b, ram0_address, ram0_data, ram0_cen_b);

// generate pixel value from reading ZBT memory
wire [7:0]    vr_pixel;
wire [18:0]  vram_addr1;
```

```
    vram_display vd1(reset,clk,hcount,vcount,vr_pixel,
            vram_addr1,vram_read_data);

    // ADV7185 NTSC decoder interface code
    // adv7185 initialization module
    adv7185init adv7185(.reset(reset), .clock_27mhz(clock_27mhz),
                    .source(1'b0), .tv_in_reset_b(tv_in_reset_b),
                    .tv_in_i2c_clock(tv_in_i2c_clock),
                    .tv_in_i2c_data(tv_in_i2c_data));

    wire [29:0] ycrcb;  // video data (luminance, chrominance)
    wire [2:0] fvh;     // sync for field, vertical, horizontal
    wire       dv;      // data valid

    ntsc_decode decode (.clk(tv_in_line_clock1), .reset(reset),
                    .tv_in_ycrcb(tv_in_ycrcb[19:10]),
                    .ycrcb(ycrcb), .f(fvh[2]),
                    .v(fvh[1]), .h(fvh[0]), .data_valid(dv));


            // /*MY CODE: CONVERT Y_CR_CB to R_G_B*/
            wire [7:0] R, G, B;
            wire [5:0] r1, g1, b1;
            wire [1:0] pix;
            wire [7:0] r_th, b_th, g_th;
            wire [7:0] r_tl, b_tl, g_tl;
    //      wire [17:0] rgb;
    //      assign r1 = R[7:2];
    //      assign g1 = G[7:2];
```

```
//      assign b1 = B[7:2];
//      assign rgb = {r1,g1,b1};
        YCrCb2RGB temp( R, G, B, clk, reset, ycrcb[29:20]/*y*/, ycrcb[19:10]/*cr*/,
ycrcb[9:0])/*cb*/;
        imageDetection imgDet(.clk(clk), .R(R),.G(G),.B(B), .pix(pix),.r_th(r_th),
.g_th(g_th), .b_th(b_th), .r_tl(r_tl), .g_tl(g_tl), .b_tl(b_tl));

        // /*END MY CODE*/

  // code to write NTSC data to video memory

  wire [18:0] ntsc_addr;
  wire [35:0] ntsc_data;
  wire        ntsc_we;
  ntsc_to_zbt n2z (clk, tv_in_line_clock1, fvh, dv, pix, //changed from ycrcb[29:22] to G
to pix
        ntsc_addr, ntsc_data, ntsc_we, 0); //changed from switch[6] to 0

  // code to write pattern to ZBT memory
  reg [31:0]   count;
  always @(posedge clk) count <= reset ? 0 : count + 1;

  wire [18:0]  vram_addr2 = count[0+18:0];
  wire [35:0]  vpat = ( switch[1] ? {4{count[3+3:3],4'b0}}
        : {4{count[3+4:4],4'b0}} );

  // mux selecting read/write to memory based on which write-enable is chosen

  wire sw_ntsc = ~switch[7];
```

```
  wire my_we = sw_ntsc ? (hcount[1:0]==2'd2) : blank;
  wire [18:0]  write_addr = ntsc_addr; //changed from: see below
  wire [35:0]  write_data = ntsc_data; //changed from: see below

//  wire [18:0]        write_addr = sw_ntsc ? ntsc_addr : vram_addr2;
//  wire [35:0]        write_data = sw_ntsc ? ntsc_data : 8'h0;//changed from vpat to 8'h0


//  wire        write_enable = sw_ntsc ? (my_we & ntsc_we) : my_we;
//  assign      vram_addr = write_enable ? write_addr : vram_addr1;
//  assign      vram_we = write_enable;

  assign      vram_addr = my_we ? write_addr : vram_addr1;
  assign      vram_we = my_we;
  assign      vram_write_data = write_data;

  // select output pixel data

  reg [7:0]     pixel = 0;
  reg   b,hs,vs;

        parameter RED = 1;
        parameter GREEN = 2;
        parameter BLUE = 3;

        reg [7:0] pix_red=0;
        reg [7:0] pix_green=0;
        reg [7:0] pix_blue=0;
```

```
        wire [2:0] threshold;
        wire [7:0] value, value_to_disp;
        assign threshold = switch[7:5];
        assign value = switch[4:0];
        assign value_to_disp = value << 3; //multiply by 8

        setThreshold setthr(.threshold(threshold), .value(value), .button(user_reset),
.r_th(r_th), .g_th(g_th), .b_th(b_th), .r_tl(r_tl), .g_tl(g_tl), .b_tl(b_tl), .sw(switch[7]));
        //user_reset is enter!!!

//    wire [9:0] h0 = 340;
//    wire [9:0] h1 = 340+227;
//    wire [9:0] h2 = 340+227+228;
//    wire [9:0] h3 = 340+227+228+228;
//
//    wire [9:0] v0 = 0;
//    wire [9:0] v1 = 255;
//    wire [9:0] v2 = 255+256;
//    wire [9:0] v3 = 255+256+256;
    wire [9:0] h0, h1, h2, h3, v0, v1, v2, v3, cur_val_axis;
    wire [2:0] cur_axis;
    wire [9:0] led1_x, led2_x, led3_x,led4_x,led5_x;
    wire [9:0]  led1_y, led2_y, led3_y, led4_y, led5_y;
    parameter low_h = 36; parameter high_h = 748;
    parameter low_v = 80; parameter high_v = 567;

    setAxis settingTheAxis(clk, left, right, down, up, enter, h0, h1, h2, h3, v0, v1, v2, v3,
cur_val_axis, cur_axis, switch[2:0], switch[7]);
```

```
    centroidLocator ctrd(reset,clk,hcount,vcount,ready,pix, h0, h1, h2, h3, v0, v1, v2, v3,
                                                            low_h, high_h, low_v, high_v,
                                                            led1_x, led2_x,
led3_x,led4_x,led5_x,
                                                            led1_y, led2_y, led3_y, led4_y,
led5_y);
    parameter side = 20;
    always @(posedge clk)
        begin
            //centroid
            //LED1
            if((led1_x - side < hcount) && (hcount< led1_x+side) &&
(led1_y-side<vcount) && (vcount <led1_y+side)) begin
                    pix_red <= 0;
                    pix_green <= 0;
                    pix_blue <= 255;
                end



            //background: RED
            else if(hcount < low_h || hcount > high_h || vcount <low_v ||
vcount>high_v) begin
                    pix_red <= 255;
                    pix_green <= 0;
                    pix_blue <= 0;
                end
```

```
                //grid separator

                else if((vcount==v0) || (vcount==v1) || (hcount==h0 && vcount>=v0))
begin
                                pix_red <= 0;
                                pix_green <= 0;
                                pix_blue <= 255;
                        end
//              else if(hcount == h0 || hcount == h1 || hcount == h2 || hcount == h3
//                      || vcount == v0 || vcount == v1 || vcount == v2 || vcount == v3)
begin
//                              pix_red <= 0;
//                              pix_green <= 0;
//                              pix_blue <= 255;
//                      end //if

                else begin
        case(vr_pixel)

                RED: begin
                pix_red <= 255;
                pix_green <= 0;
                pix_blue <= 0;
                end
                GREEN: begin
                pix_red <= 0;
                pix_green <= 255;
                pix_blue <= 0;
                end
```

```
            //actually yellow but...
            BLUE: begin
            pix_red <= 255;
            pix_green <= 255;
            pix_blue <= 0;
            end

            default: begin
            pix_red <= 0;
            pix_green <= 0;
            pix_blue <= 0;
            end

      endcase
            end //else i.e. not part of the grid
      pixel <= vr_pixel; //switched from pixel <= switch[0] ? {hcount[8:6],5'b0} : vr_pixel;
      b <= blank;
      hs <= hsync;
      vs <= vsync;
      end //end always



// VGA Output.  In order to meet the setup and hold times of the
// AD7125, we send it ~clk.
assign vga_out_red = pix_red; //changed from pixel
assign vga_out_green = pix_green; //changed from pixel
assign vga_out_blue = pix_blue; //changed from pixel
assign vga_out_sync_b = 1'b1;  // not used
```

```
  assign vga_out_pixel_clock = ~clk;
  assign vga_out_blank_b = ~b;
  assign vga_out_hsync = hs;
  assign vga_out_vsync = vs;



  // debugging

  assign led = ~{vram_addr[18:13],reset,switch[0]};

  always @(posedge clk)
        // dispdata <= {vram_read_data,9'b0,vram_addr};
//      dispdata <= {ntsc_data,9'b0,ntsc_addr};
//      dispdata <= 64'hffff_ffff_ffff_ffff;
//      dispdata <= {value_to_disp,8'h00,r_th, b_th, g_th, r_tl, b_tl, g_tl};
         //[2:0]cur_axis, [9:0] cur_val_axis
//       dispdata <= {0,cur_axis,2'b0 ,cur_val_axis};
//       dispdata <= {0,switch[2:0],2'b0 ,cur_val_axis};
         dispdata <= {0,led1_y, 2'b0 ,led1_x};
endmodule


//////////////////////////////////////////////////////////////////////////////////
// imageDetection: outputs whether there is RED, GREEN, or BLUE (YELLOW??) color
by thresholding
//
module imageDetection(input clk, input [7:0] R,G,B, r_th, g_th, b_th, r_tl, g_tl, b_tl ,
output reg [1:0] pix);
```

```
//      wire [7:0] R_TH, R_TL, B_TH, B_TL, G_TH, G_TL;
//
//      //HIGH thresholds
//      assign R_TH = 8'd180; //200 is fine
//      assign G_TH = 8'd180;
//      assign B_TH = 8'd180;
//
//      //LOW thresholds
//      assign R_TL = 8'd100;
//      assign G_TL = 8'd100;
//      assign B_TL = 8'd100;


        //parameters
        parameter RED = 1;
        parameter GREEN = 2;
        parameter BLUE = 3; //yellow = red + green

        //you know
        always@(posedge clk) begin
//      if(R>R_TH && B<B_TL && G < G_TL) pix <= RED;
//      else if(R<R_TL && B<B_TL && G > G_TH) pix <= GREEN;

//      if(R<R_TL && B>B_TH && G < G_TL) pix <= BLUE;

//      if(B>B_TH) pix<=BLUE;

        //ignores bright lights
```

```
//        if(R>R_TH && B>B_TH && G>B_TH) pix <= 0;
          if(R>r_th&& B<b_tl && G < g_tl) pix <= RED;
          else if(R<r_tl && B<b_tl && G > g_th) pix <= GREEN;
          else if (R>r_th && B<b_tl&& G > g_th) pix <= BLUE; //but actually yellow
//        if(R>R_TH) pix <=RED;
//        else if(G>G_TH) pix <=GREEN;
//        else if(B>B_TH) pix <=BLUE;
          else pix <= 0;
          end //always
endmodule


module setAxis(input clk, left, right, down, up, enter, output reg [9:0] h0, h1, h2, h3,
output reg[9:0] v0, v1, v2, v3, output reg[9:0] cur_val_axis, input [2:0] cur_axis,
input[2:0] sw2,input sw);
    parameter x0 = 0; parameter x1 = 1; parameter x2 = 2; parameter x3 = 3;
          parameter y0 = 4; parameter y1 = 5; parameter y2 = 6; parameter y3 = 7;
    parameter s0 = 0; parameter s1 = 1; parameter s2 = 2; parameter s3 = 3;
    parameter s4 = 4; parameter step = 10;
    reg[1:0] state ; reg[2:0] old_switch;
    initial begin
          cur_val_axis <= 10'h74;
          state <= 0;
          h0 <= 10'h18b;
          v0 <= 10'hd9;
          v1 <= 10'h1a1;
//        h0 <= 10'h74;//340;
//        h1 <= 10'hf7;//340+227;
//        h2 <= 10'h18b;//340+227+228;
```

```
//          h3 <= 10'h247;//340+227+228+228;
//          v0 <= 10'h78;//0;
//          v1 <= 10'h89;//255;
//          v2 <= 10'h119;//255+256;
//          v3 <= 10'h1a1;//255+256+256;
            old_switch <= 0;
    end //initial



    always@(posedge clk) begin
            if(sw) begin
                    if(old_switch != sw2[2:0]) begin
                            case(sw2[2:0])
                                    x0: cur_val_axis <= h0;
                                    x1: cur_val_axis <= h1;
                                    x2: cur_val_axis <= h2;
                                    x3: cur_val_axis <= h3;
                                    y0: cur_val_axis <= v0;
                                    y1: cur_val_axis <= v1;
                                    y2: cur_val_axis <= v2;
                                    y3: cur_val_axis <= v3;
                            endcase
                            old_switch <= sw2[2:0];
                    end //if old switch

                    else begin
                            case(state)
                                    s0: begin
```

```
                        if(down) begin
                                cur_val_axis <= cur_val_axis-step;
                                state <= s1;
                                end
                        else if(up) begin
                                cur_val_axis <= cur_val_axis + step;
                                state <= s1;
                                end
                        end //s0
                s1: begin
                        if(!up && !down) state <= s2;
                        end //s1
                s2: begin
                        case(sw2[2:0])
                                x0: h0 <= cur_val_axis;
                                x1: h1 <= cur_val_axis;
                                x2: h2 <= cur_val_axis;
                                x3: h3 <= cur_val_axis;
                                y0: v0 <= cur_val_axis;
                                y1: v1 <= cur_val_axis;
                                y2: v2 <= cur_val_axis;
                                y3: v3 <= cur_val_axis;
                        endcase
                        state <= 0;
                        end
        endcase


end //else
```

```
        end //~sw
    end //always

endmodule

module setThreshold(input [2:0] threshold, input[7:0] value, input button, output reg [7:0]
r_th, g_th, b_th, r_tl, g_tl, b_tl, input sw);

        reg done = 0;
        reg [7:0] thresh_value = 0;
        initial begin
        r_th <= 180;
        b_th <= 180;
        g_th <= 180;

        r_tl <= 100;
        b_tl <= 100;
        g_tl <= 100;
        end//

        always@(posedge button) begin
         if(~sw) begin
        if(!done) begin
        done <= ~done;
        thresh_value <= value <<3;
        end //if

        else begin
```

```
        case(threshold)
                0: r_th <= thresh_value;
                1: g_th <= thresh_value;
                2: b_th <= thresh_value;
                3: r_tl <= thresh_value;
                4: g_tl <= thresh_value;
                5: b_tl <= thresh_value;
                default: r_th <= r_th;
        endcase
        done <= ~done;
        end //else
         end //sw
        end //always
endmodule

module centroidLocator(reset,clk,hcount,vcount,ready,pix, h0, h1, h2, h3, v0, v1, v2, v3,
                                                        low_h, high_h, low_v, high_v,
                                                        led1_x, led2_x,
led3_x,led4_x,led5_x,
                                                        led1_y, led2_y, led3_y, led4_y,
led5_y
                                                        );
    input reset, clk;
   input [10:0] hcount;
   input [9:0]  vcount;
    output ready;
    input [1:0] pix;
    input [9:0] h0, h1, h2, h3, v0, v1, v2, v3, low_h, high_h, low_v, high_v;
    output [10:0] led1_x, led2_x, led3_x,led4_x,led5_x;
```

```
output [9:0]  led1_y, led2_y, led3_y, led4_y, led5_y;

parameter RED = 1;
parameter GREEN = 2;
parameter BLUE = 3; //yellow = red + green

//                    L1
//
//    L2                    L3
//
//    L4                    L5

//running sum of the coordinates
reg [17:0] h_sum_1, h_sum_2, h_sum_3, h_sum_4, h_sum_5;
reg [19:0] v_sum_1, v_sum_2, v_sum_3, v_sum_4, v_sum_5;

//samples of x and y coordinate thus far; you take the average of these samples to
find x,y of each of the LEDs
reg [7:0] h_samples_1 [7:0]; //need for other LEDs as well
reg [7:0] v_samples_1 [7:0]; //need for other LEDs as well

//num of samples that have been taken for x/y coor
reg [5:0] counts_h1; //need for other LEDs as well
reg [7:0] counts_v1; //need for other LEDs as well

parameter WIDTH = 10;
wire [WIDTH-1:0] temp_x1, temp_y1, ready_x1, ready_y1;
wire [WIDTH-1:0] remainder;
//GRID: so that its easier to find average in that section
```

```
//    assign led1_x = temp_x1;
//    assign led1_y = temp_y1;

   assign led1_x = (h_samples_1[0] + h_samples_1[1] + h_samples_1[2]+
h_samples_1[3]
                                                + h_samples_1[4] + h_samples_1[5] +
h_samples_1[6] + h_samples_1[7]) >> 3;
   //similarly for  led2_x, etc
   assign led1_y = (v_samples_1[0] + v_samples_1[1] + v_samples_1[2]+
v_samples_1[3]
                                                + v_samples_1[4] + v_samples_1[5] +
v_samples_1[6] + v_samples_1[7]) >> 3;
   //similarly for  led2_y, etc

   assign temp_x1 = (h_sum_1)>>7;
   assign temp_y1 = (v_sum_1)>>7;


   initial begin
         h_samples_1[0] <= 10'h18b;
         h_samples_1[1] <= 10'h18b;
         h_samples_1[2] <= 10'h18b;
         h_samples_1[3] <= 10'h18b;
         h_samples_1[4] <= 10'h18b;
         h_samples_1[5] <= 10'h18b;
         h_samples_1[6] <= 10'h18b;
         h_samples_1[7] <= 10'h18b;

         v_samples_1[0] <= 10'hd0;
```

```
            v_samples_1[1] <= 10'hd0;
            v_samples_1[2] <= 10'hd0;
            v_samples_1[3] <= 10'hd0;
            v_samples_1[4] <= 10'hd0;
            v_samples_1[5] <= 10'hd0;
            v_samples_1[6] <= 10'hd0;
            v_samples_1[7] <= 10'hd0;

    end
    reg[2:0] square=0;
    ////////////////////v0
    ///    ///1    ///    ///
    ///    ///    ///    ///
    ////////////////////v1
    ///2    ///    ///3    ///
    ///    ///    ///    ///
    ////////////////////v2
    ///4    ///    ///5    ///
    ///    ///    ///    ///
    ////////////////////v3
//    h0            h1       h2       h3

    //determines which square pixel is in
    always @(posedge clk) begin
//          if(h1<=hcount && hcount<=h2 && v0 <= vcount && vcount <= v1) square <= 1;
//          else if(h0<=hcount && hcount<=h1 && v1 <= vcount && vcount <= v2) square
<= 2;
//          else if(h2<=hcount && hcount<=h3 && v1 <= vcount && vcount <= v2) square
<= 3;
```

```
//        else if(h0<=hcount && hcount<=h1 && v2 <= vcount && vcount <= v3) square
<= 4;
//        else if(h2<=hcount && hcount<=h3 && v2 <= vcount && vcount <= v3) square
<= 5;
        if(vcount<v0) square <=1;
        else if(vcount <= v1) begin
                if(hcount<h0) square <= 2;
                else square <= 3;
                end//else if
        else if(vcount>v1)    begin
                if(hcount<h0) square <= 4;
                else square <= 5;
                end//else if
        else square <= 0;
    end//always




//    divider #(WIDTH) x1(clk, sign, start, h_sum_1, counts_h1, temp_x1, remainder,
ready_x1);
//    divider #(WIDTH) y1(clk, sign, start, v_sum_1, counts_v1, temp_y1, remainder,
ready_y1);

    reg [3:0] i,j;

    parameter counting_times = 128;
```

```
    reg[4:0] led1_count;
    wire [1:0] color;
    assign color = pix;
    reg r;
    always @(posedge clk) begin
        //only issue is figuring out the color!!!! should it be done at the same time???
        if((low_h<=hcount) && (hcount<= high_h) && (low_v<=vcount) &&
(vcount<=high_v)) begin //if it's in the image
            /*insert code to determine color either simultaneously or with LUT i.e. ZBT
here */
            case(square)
                1: if((color==GREEN) && (led1_count<counting_times))begin
                        h_sum_1 <= h_sum_1 + hcount;
                        v_sum_1 <= v_sum_1 + vcount;
                        led1_count <= led1_count+1;
                    end //
//              2: if(color==GREEN)begin
//                      h_sum_2 <= h_sum_2 + hcount;
//                      v_sum_2 <= v_sum_2 + vcount;
//                  end //
//              3: if(color==GREEN)begin
//                      h_sum_3 <= h_sum_3 + hcount;
//                      v_sum_3 <= v_sum_3 + vcount;
//                  end //
//              4: if(color==GREEN)begin
//                      h_sum_4 <= h_sum_4 + hcount;
//                      v_sum_4 <= v_sum_4 + vcount;
//                  end //
//              5: if(color==GREEN)begin
```

```
//                                      h_sum_5 <= h_sum_5 + hcount;
//                                      v_sum_5 <= v_sum_5 + vcount;
//                                  end //
                        default: r<=r; //NOP
                endcase

        end //if pixel at hcount, vcount is in the main image

        else if(hcount==1000 && vcount==700) begin //time to start calculating centroid
and add new value to array!
                h_samples_1[0] <= temp_x1;
                v_samples_1[0] <= temp_y1;
                for(i=1; i<=7; i=i+1) begin
                        h_samples_1[i] <= h_samples_1[i-1];
                        v_samples_1[i] <= v_samples_1[i-1];
                end
//              temp_x1 <= h_sum_1 >>5;
//              temp_y1 <= v_sum_1 >>5;
                //also reset
                h_sum_1 <= 0; v_sum_1<=0; counts_h1<=0; led1_count <= 0; //should
do for LEDs as well
        end//if hcount, vcount = 1024,768


    end //always

endmodule
```

```
//////////////////////////////////////////////////////////////////////////////
// xvga: Generate XVGA display signals (1024 x 768 @ 60Hz)

module xvga(vclock,hcount,vcount,hsync,vsync,blank);
  input vclock;
  output [10:0] hcount;
  output [9:0] vcount;
  output       vsync;
  output       hsync;
  output       blank;

  reg          hsync,vsync,hblank,vblank,blank;
  reg [10:0]   hcount;         // pixel number on current line
  reg [9:0] vcount;    // line number

  // horizontal: 1344 pixels total
  // display 1024 pixels per line
  wire         hsyncon,hsyncoff,hreset,hblankon;
  assign       hblankon = (hcount == 1023);
  assign       hsyncon = (hcount == 1047);
  assign       hsyncoff = (hcount == 1183);
  assign       hreset = (hcount == 1343);

  // vertical: 806 lines total
  // display 768 lines
  wire         vsyncon,vsyncoff,vreset,vblankon;
  assign       vblankon = hreset & (vcount == 767);
  assign       vsyncon = hreset & (vcount == 776);
```

```
  assign       vsyncoff = hreset & (vcount == 782);
  assign       vreset = hreset & (vcount == 805);

  // sync and blanking
  wire         next_hblank,next_vblank;
  assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
  assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;
  always @(posedge vclock) begin
      hcount <= hreset ? 0 : hcount + 1;
      hblank <= next_hblank;
      hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync;  // active low

      vcount <= hreset ? (vreset ? 0 : vcount + 1) : vcount;
      vblank <= next_vblank;
      vsync <= vsyncon ? 0 : vsyncoff ? 1 : vsync;  // active low

      blank <= next_vblank | (next_hblank & ~hreset);
  end
endmodule

/*
////////////////////////////////////////////////////////////////////////////////
// xvga: Generate XVGA display signals (800 x 600 @ 60Hz)

module xvga(vclock,hcount,vcount,hsync,vsync,blank);
  input vclock;
  output [10:0] hcount;
  output [9:0] vcount;
  output       vsync;
```

```
output      hsync;
output      blank;

reg         hsync,vsync,hblank,vblank,blank;
reg [10:0]  hcount;       // pixel number on current line
reg [9:0] vcount;   // line number

// horizontal: 1056 pixels total
// display 800 pixels per line
wire        hsyncon,hsyncoff,hreset,hblankon;
assign      hblankon = (hcount == 799);
assign      hsyncon = (hcount == 839);
assign      hsyncoff = (hcount == 967);
assign      hreset = (hcount == 1055);

// vertical: 628 lines total
// display 600 lines
wire        vsyncon,vsyncoff,vreset,vblankon;
assign      vblankon = hreset & (vcount == 599);
assign      vsyncon = hreset & (vcount == 600);
assign      vsyncoff = hreset & (vcount == 604);
assign      vreset = hreset & (vcount == 627);

// sync and blanking
wire        next_hblank,next_vblank;
assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;
always @(posedge vclock) begin
      hcount <= hreset ? 0 : hcount + 1;
```

```
        hblank <= next_hblank;
        hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync;  // active low

        vcount <= hreset ? (vreset ? 0 : vcount + 1) : vcount;
        vblank <= next_vblank;
        vsync <= vsyncon ? 0 : vsyncoff ? 1 : vsync;  // active low

        blank <= next_vblank | (next_hblank & ~hreset);
    end
endmodule */

//////////////////////////////////////////////////////////////////////////
// generate display pixels from reading the ZBT ram
// note that the ZBT ram has 2 cycles of read (and write) latency
//
// We take care of that by latching the data at an appropriate time.
//
// Note that the ZBT stores 36 bits per word; we use only 32 bits here,
// decoded into four bytes of pixel data.
//
// Bug due to memory management will be fixed. The bug happens because
// memory is called based on current hcount & vcount, which will actually
// shows up 2 cycle in the future. Not to mention that these incoming data
// are latched for 2 cycles before they are used. Also remember that the
// ntsc2zbt's addressing protocol has been fixed.

// The original bug:
// -. At (hcount, vcount) = (100, 201) data at memory address(0,100,49)
//         arrives at vram_read_data, latch it to vr_data_latched.
```

```
// -. At (hcount, vcount) = (100, 203) data at memory address(0,100,49)
//        is latched to last_vr_data to be used for display.
// -. Remember that memory address(0,100,49) contains camera data
//        pixel(100,192) - pixel(100,195).
// -. At (hcount, vcount) = (100, 204) camera pixel data(100,192) is shown.
// -. At (hcount, vcount) = (100, 205) camera pixel data(100,193) is shown.
// -. At (hcount, vcount) = (100, 206) camera pixel data(100,194) is shown.
// -. At (hcount, vcount) = (100, 207) camera pixel data(100,195) is shown.
//
// Unfortunately this means that at (hcount == 0) to (hcount == 11) data from
// the right side of the camera is shown instead (including possible sync signals).

// To fix this, two corrections has been made:
// -. Fix addressing protocol in ntsc_to_zbt module.
// -. Forecast hcount & vcount 8 clock cycles ahead and use that
//        instead to call data from ZBT.


module vram_display(reset,clk,hcount,vcount,vr_pixel,
        vram_addr,vram_read_data);

  input reset, clk;
  input [10:0] hcount;
  input [9:0]   vcount;
  output [7:0] vr_pixel;
  output [18:0] vram_addr;
  input [35:0]  vram_read_data;

  //forecast hcount & vcount 8 clock cycles ahead to get data from ZBT
```

```
  wire [10:0] hcount_f = (hcount >= 1048) ? (hcount - 1048) : (hcount + 8);
  wire [9:0] vcount_f = (hcount >= 1048) ? ((vcount == 805) ? 0 : vcount + 1) : vcount;

  wire [18:0]          vram_addr = {1'b0, vcount_f, hcount_f[9:2]};

  wire [1:0]    hc4 = hcount[1:0];
  reg [7:0]            vr_pixel;
  reg [35:0]    vr_data_latched;
  reg [35:0]    last_vr_data;

  always @(posedge clk)
      last_vr_data <= (hc4==2'd3) ? vr_data_latched : last_vr_data;

  always @(posedge clk)
      vr_data_latched <= (hc4==2'd1) ? vram_read_data : vr_data_latched;

  always @(*)       // each 36-bit word from RAM is decoded to 4 bytes
      case (hc4)
      2'd3: vr_pixel = last_vr_data[7:0];
      2'd2: vr_pixel = last_vr_data[7+8:0+8];
      2'd1: vr_pixel = last_vr_data[7+16:0+16];
      2'd0: vr_pixel = last_vr_data[7+24:0+24];
      endcase

endmodule // vram_display

////////////////////////////////////////////////////////////////////////////////
// parameterized delay line
```

```
module delayN(clk,in,out);
  input clk;
  input in;
  output out;

  parameter NDELAY = 3;

  reg [NDELAY-1:0] shiftreg;
  wire             out = shiftreg[NDELAY-1];

  always @(posedge clk)
        shiftreg <= {shiftreg[NDELAY-2:0],in};

endmodule // delayN

//////////////////////////////////////////////////////////////////////////////
// ramclock module

//////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- ZBT RAM clock generation
//
//
// Created: April 27, 2004
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////
//
// This module generates deskewed clocks for driving the ZBT SRAMs and FPGA
```

```
// registers. A special feedback trace on the labkit PCB (which is length
// matched to the RAM traces) is used to adjust the RAM clock phase so that
// rising clock edges reach the RAMs at exactly the same time as rising clock
// edges reach the registers in the FPGA.
//
// The RAM clock signals are driven by DDR output buffers, which further
// ensures that the clock-to-pad delay is the same for the RAM clocks as it is
// for any other registered RAM signal.
//
// When the FPGA is configured, the DCMs are enabled before the chip-level I/O
// drivers are released from tristate. It is therefore necessary to
// artificially hold the DCMs in reset for a few cycles after configuration.
// This is done using a 16-bit shift register. When the DCMs have locked, the
// <lock> output of this mnodule will go high. Until the DCMs are locked, the
// ouput clock timings are not guaranteed, so any logic driven by the
// <fpga_clock> should probably be held inreset until <locked> is high.
//
//////////////////////////////////////////////////////////////////////////////

module ramclock(ref_clock, fpga_clock, ram0_clock, ram1_clock,
        clock_feedback_in, clock_feedback_out, locked);

  input ref_clock;              // Reference clock input
  output fpga_clock;                    // Output clock to drive FPGA logic
  output ram0_clock, ram1_clock;   // Output clocks for each RAM chip
  input  clock_feedback_in;         // Output to feedback trace
  output clock_feedback_out;       // Input from feedback trace
  output locked;                // Indicates that clock outputs are stable
```

```
wire  ref_clk, fpga_clk, ram_clk, fb_clk, lock1, lock2, dcm_reset;

//////////////////////////////////////////////////////////////////////////////

//To force ISE to compile the ramclock, this line has to be removed.
//IBUFG ref_buf (.O(ref_clk), .I(ref_clock));

      assign ref_clk = ref_clock;

BUFG int_buf (.O(fpga_clock), .I(fpga_clk));

DCM int_dcm (.CLKFB(fpga_clock),
      .CLKIN(ref_clk),
      .RST(dcm_reset),
      .CLK0(fpga_clk),
      .LOCKED(lock1));
// synthesis attribute DLL_FREQUENCY_MODE of int_dcm is "LOW"
// synthesis attribute DUTY_CYCLE_CORRECTION of int_dcm is "TRUE"
// synthesis attribute STARTUP_WAIT of int_dcm is "FALSE"
// synthesis attribute DFS_FREQUENCY_MODE of int_dcm is "LOW"
// synthesis attribute CLK_FEEDBACK of int_dcm  is "1X"
// synthesis attribute CLKOUT_PHASE_SHIFT of int_dcm is "NONE"
// synthesis attribute PHASE_SHIFT of int_dcm is 0

BUFG ext_buf (.O(ram_clock), .I(ram_clk));

IBUFG fb_buf (.O(fb_clk), .I(clock_feedback_in));

DCM ext_dcm (.CLKFB(fb_clk),
```

```
        .CLKIN(ref_clk),
        .RST(dcm_reset),
        .CLK0(ram_clk),
        .LOCKED(lock2));
  // synthesis attribute DLL_FREQUENCY_MODE of ext_dcm is "LOW"
  // synthesis attribute DUTY_CYCLE_CORRECTION of ext_dcm is "TRUE"
  // synthesis attribute STARTUP_WAIT of ext_dcm is "FALSE"
  // synthesis attribute DFS_FREQUENCY_MODE of ext_dcm is "LOW"
  // synthesis attribute CLK_FEEDBACK of ext_dcm  is "1X"
  // synthesis attribute CLKOUT_PHASE_SHIFT of ext_dcm is "NONE"
  // synthesis attribute PHASE_SHIFT of ext_dcm is 0

  SRL16 dcm_rst_sr (.D(1'b0), .CLK(ref_clk), .Q(dcm_reset),
        .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
  // synthesis attribute init of dcm_rst_sr is "000F";


  OFDDRRSE ddr_reg0 (.Q(ram0_clock), .C0(ram_clock), .C1(~ram_clock),
        .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));
  OFDDRRSE ddr_reg1 (.Q(ram1_clock), .C0(ram_clock), .C1(~ram_clock),
        .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));
  OFDDRRSE ddr_reg2 (.Q(clock_feedback_out), .C0(ram_clock), .C1(~ram_clock),
        .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));

  assign locked = lock1 && lock2;

endmodule
```

zbt_6111.v

```
//
// File:   zbt_6111.v
// Date:   27-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Simple ZBT driver for the MIT 6.111 labkit, which does not hide the
// pipeline delays of the ZBT from the user.  The ZBT memories have
// two cycle latencies on read and write, and also need extra-long data hold
// times around the clock positive edge to work reliably.
//

//////////////////////////////////////////////////////////////////////////////////
// Ike's simple ZBT RAM driver for the MIT 6.111 labkit
//
// Data for writes can be presented and clocked in immediately; the actual
// writing to RAM will happen two cycles later.
//
// Read requests are processed immediately, but the read data is not available
// until two cycles after the intial request.
//
// A clock enable signal is provided; it enables the RAM clock when high.

module zbt_6111(clk, cen, we, addr, write_data, read_data,
                ram_clk, ram_we_b, ram_address, ram_data, ram_cen_b);

  input clk;                // system clock
  input cen;                        // clock enable for gating ZBT cycles
  input we;                 // write enable (active HIGH)
```

```
input [18:0] addr;              // memory address
input [35:0] write_data;    // data to write
output [35:0] read_data;    // data read from memory
output      ram_clk;    // physical line to ram clock
output      ram_we_b;    // physical line to ram we_b
output [18:0] ram_address;    // physical line to ram address
inout [35:0]  ram_data;    // physical line to ram data
output      ram_cen_b;    // physical line to ram clock enable

// clock enable (should be synchronous and one cycle high at a time)
wire      ram_cen_b = ~cen;

// create delayed ram_we signal: note the delay is by two cycles!
// ie we present the data to be written two cycles after we is raised
// this means the bus is tri-stated two cycles after we is raised.

reg [1:0]   we_delay;

always @(posedge clk)
      we_delay <= cen ? {we_delay[0],we} : we_delay;

// create two-stage pipeline for write data

reg [35:0]  write_data_old1;
reg [35:0]  write_data_old2;
always @(posedge clk)
      if (cen)
      {write_data_old2, write_data_old1} <= {write_data_old1, write_data};
```

```
  // wire to ZBT RAM signals

  assign       ram_we_b = ~we;
  assign       ram_clk = 1'b0;  // gph 2011-Nov-10
                        // set to zero as place holder

//   assign     ram_clk = ~clk;        // RAM is not happy with our data hold
                        // times if its clk edges equal FPGA's
                        // so we clock it on the falling edges
                        // and thus let data stabilize longer
  assign       ram_address = addr;

  assign       ram_data = we_delay[1] ? write_data_old2 : {36{1'bZ}};
  assign       read_data = ram_data;

endmodule // zbt_6111
```

ycbcr2rgb.v

```
/////////////////////////////////////////////////////////////////////
 **
 ** Module: ycrcb2rgb
 **
 ** Generic Equations:
 /////////////////////////////////////////////////////////////////////*/

module YCrCb2RGB ( R, G, B, clk, rst, Y, Cr, Cb); //changed from: added after Cb
variable(s) pix

output [7:0]  R, G, B;
```

```
input clk,rst;
input[9:0] Y, Cr, Cb;

wire [7:0] R,G,B;
reg [20:0] R_int,G_int,B_int,X_int,A_int,B1_int,B2_int,C_int;
reg [9:0] const1,const2,const3,const4,const5;
reg[9:0] Y_reg, Cr_reg, Cb_reg;

//registering constants
always @ (posedge clk)
begin
 const1 = 10'b 0100101010; //1.164 = 01.00101010
 const2 = 10'b 0110011000; //1.596 = 01.10011000
 const3 = 10'b 0011010000; //0.813 = 00.11010000
 const4 = 10'b 0001100100; //0.392 = 00.01100100
 const5 = 10'b 1000000100; //2.017 = 10.00000100
end

always @ (posedge clk or posedge rst)
  if (rst)
        begin
        Y_reg <= 0; Cr_reg <= 0; Cb_reg <= 0;
        end
  else
        begin
     Y_reg <= Y; Cr_reg <= Cr; Cb_reg <= Cb;
        end
```

```
always @ (posedge clk or posedge rst)
  if (rst)
        begin
        A_int <= 0; B1_int <= 0; B2_int <= 0; C_int <= 0; X_int <= 0;
        end
  else
        begin
        X_int <= (const1 * (Y_reg - 'd64)) ;
        A_int <= (const2 * (Cr_reg - 'd512));
        B1_int <= (const3 * (Cr_reg - 'd512));
        B2_int <= (const4 * (Cb_reg - 'd512));
        C_int <= (const5 * (Cb_reg - 'd512));
        end

always @ (posedge clk or posedge rst)
  if (rst)
        begin
        R_int <= 0; G_int <= 0; B_int <= 0;
        end
  else
        begin
        R_int <= X_int + A_int;
        G_int <= X_int - B1_int - B2_int;
        B_int <= X_int + C_int;
        end



/*always @ (posedge clk or posedge rst)
```

```
  if (rst)
        begin
        R_int <= 0; G_int <= 0; B_int <= 0;
        end
  else
        begin
        X_int <= (const1 * (Y_reg - 'd64)) ;
        R_int <= X_int + (const2 * (Cr_reg - 'd512));
        G_int <= X_int - (const3 * (Cr_reg - 'd512)) - (const4 * (Cb_reg - 'd512));
        B_int <= X_int + (const5 * (Cb_reg - 'd512));
        end

*/
/* limit output to 0 - 4095, <0 equals 0 and >4095 equals 4095 */
///^^^ a typo??? output is only 8 bits so upper limit is 8'b1111_1111= 255
assign R = (R_int[20]) ? 0 : (R_int[19:18] == 2'b0) ? R_int[17:10] : 8'b11111111;
assign G = (G_int[20]) ? 0 : (G_int[19:18] == 2'b0) ? G_int[17:10] : 8'b11111111;
assign B = (B_int[20]) ? 0 : (B_int[19:18] == 2'b0) ? B_int[17:10] : 8'b11111111;

endmodule
```

video_decoder.v

```
//
// File:   video_decoder.v
// Date:   31-Oct-05
// Author: J. Castro (MIT 6.111, fall 2005)
//
// This file contains the ntsc_decode and adv7185init modules
//
```

```
// These modules are used to grab input NTSC video data from the RCA
// phono jack on the right hand side of the 6.111 labkit (connect
// the camera to the LOWER jack).
//

//////////////////////////////////////////////////////////////////////////
//
// NTSC decode - 16-bit CCIR656 decoder
// By Javier Castro
// This module takes a stream of LLC data from the adv7185
// NTSC/PAL video decoder and generates the corresponding pixels,
// that are encoded within the stream, in YCrCb format.

// Make sure that the adv7185 is set to run in 16-bit LLC2 mode.

module ntsc_decode(clk, reset, tv_in_ycrcb, ycrcb, f, v, h, data_valid);

  // clk - line-locked clock (in this case, LLC1 which runs at 27Mhz)
  // reset - system reset
  // tv_in_ycrcb - 10-bit input from chip. should map to pins [19:10]
  // ycrcb - 24 bit luminance and chrominance (8 bits each)
  // f - field: 1 indicates an even field, 0 an odd field
  // v - vertical sync: 1 means vertical sync
  // h - horizontal sync: 1 means horizontal sync

  input clk;
  input reset;
  input [9:0] tv_in_ycrcb; // modified for 10 bit input - should be P[19:10]
  output [29:0] ycrcb;
```

```
output     f;
output     v;
output     h;
output     data_valid;
// output [4:0] state;

parameter     SYNC_1 = 0;
parameter     SYNC_2 = 1;
parameter     SYNC_3 = 2;
parameter     SAV_f1_cb0 = 3;
parameter     SAV_f1_y0 = 4;
parameter     SAV_f1_cr1 = 5;
parameter     SAV_f1_y1 = 6;
parameter     EAV_f1 = 7;
parameter     SAV_VBI_f1 = 8;
parameter     EAV_VBI_f1 = 9;
parameter     SAV_f2_cb0 = 10;
parameter     SAV_f2_y0 = 11;
parameter     SAV_f2_cr1 = 12;
parameter     SAV_f2_y1 = 13;
parameter     EAV_f2 = 14;
parameter     SAV_VBI_f2 = 15;
parameter     EAV_VBI_f2 = 16;



// In the start state, the module doesn't know where
// in the sequence of pixels, it is looking.
```

```
  // Once we determine where to start, the FSM goes through a normal
  // sequence of SAV process_YCrCb EAV... repeat

  // The data stream looks as follows
  // SAV_FF | SAV_00 | SAV_00 | SAV_XY | Cb0 | Y0 | Cr1 | Y1 | Cb2 | Y2 | ... | EAV
sequence
  // There are two things we need to do:
  //   1. Find the two SAV blocks (stands for Start Active Video perhaps?)
  //   2. Decode the subsequent data

  reg [4:0]     current_state = 5'h00;
  reg [9:0]     y = 10'h000;  // luminance
  reg [9:0]     cr = 10'h000; // chrominance
  reg [9:0]     cb = 10'h000; // more chrominance

  assign     state = current_state;

  always @ (posedge clk)
        begin
   if (reset)
     begin

     end
   else
     begin
        // these states don't do much except allow us to know where we are in the
stream.
        // whenever the synchronization code is seen, go back to the sync_state before
```

```
// transitioning to the new state
case (current_state)
SYNC_1: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_2 : SYNC_1;
SYNC_2: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_3 : SYNC_1;
SYNC_3: current_state <= (tv_in_ycrcb == 10'h200) ? SAV_f1_cb0 :
                         (tv_in_ycrcb == 10'h274) ? EAV_f1 :
                         (tv_in_ycrcb == 10'h2ac) ? SAV_VBI_f1 :
                         (tv_in_ycrcb == 10'h2d8) ? EAV_VBI_f1 :
                         (tv_in_ycrcb == 10'h31c) ? SAV_f2_cb0 :
                         (tv_in_ycrcb == 10'h368) ? EAV_f2 :
                         (tv_in_ycrcb == 10'h3b0) ? SAV_VBI_f2 :
                         (tv_in_ycrcb == 10'h3c4) ? EAV_VBI_f2 : SYNC_1;

SAV_f1_cb0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_y0;
SAV_f1_y0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_cr1;
SAV_f1_cr1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_y1;
SAV_f1_y1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_cb0;

SAV_f2_cb0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f2_y0;
SAV_f2_y0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f2_cr1;
SAV_f2_cr1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f2_y1;
SAV_f2_y1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f2_cb0;

// These states are here in the event that we want to cover these signals
// in the future. For now, they just send the state machine back to SYNC_1
EAV_f1: current_state <= SYNC_1;
SAV_VBI_f1: current_state <= SYNC_1;
EAV_VBI_f1: current_state <= SYNC_1;
EAV_f2: current_state <= SYNC_1;
```

```
        SAV_VBI_f2: current_state <= SYNC_1;
        EAV_VBI_f2: current_state <= SYNC_1;

        endcase
    end
        end // always @ (posedge clk)

// implement our decoding mechanism

wire y_enable;
wire cr_enable;
wire cb_enable;

// if y is coming in, enable the register
// likewise for cr and cb
assign y_enable = (current_state == SAV_f1_y0) ||
                  (current_state == SAV_f1_y1) ||
                  (current_state == SAV_f2_y0) ||
                  (current_state == SAV_f2_y1);
assign cr_enable = (current_state == SAV_f1_cr1) ||
                  (current_state == SAV_f2_cr1);
assign cb_enable = (current_state == SAV_f1_cb0) ||
                  (current_state == SAV_f2_cb0);

// f, v, and h only go high when active
assign {v,h} = (current_state == SYNC_3) ? tv_in_ycrcb[7:6] : 2'b00;

// data is valid when we have all three values: y, cr, cb
assign data_valid = y_enable;
```

```
  assign ycrcb = {y,cr,cb};

   /////convert to RGB, threshold here?

  reg      f = 0;

  always @ (posedge clk)
        begin
   y <= y_enable ? tv_in_ycrcb : y;
   cr <= cr_enable ? tv_in_ycrcb : cr;
   cb <= cb_enable ? tv_in_ycrcb : cb;
         f <= (current_state == SYNC_3) ? tv_in_ycrcb[8] : f;
        end




endmodule



//////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- ADV7185 Video Decoder Configuration Init
//
// Created:
// Author: Nathan Ickes
//
```

```
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
// Register 0
////////////////////////////////////////////////////////////////////////////////

`define INPUT_SELECT                    4'h0
  // 0: CVBS on AIN1 (composite video in)
  // 7: Y on AIN2, C on AIN5 (s-video in)
  // (These are the only configurations supported by the 6.111 labkit hardware)
`define INPUT_MODE                      4'h0
  // 0: Autodetect: NTSC or PAL (BGHID), w/o pedestal
  // 1: Autodetect: NTSC or PAL (BGHID), w/pedestal
  // 2: Autodetect: NTSC or PAL (N), w/o pedestal
  // 3: Autodetect: NTSC or PAL (N), w/pedestal
  // 4: NTSC w/o pedestal
  // 5: NTSC w/pedestal
  // 6: NTSC 4.43 w/o pedestal
  // 7: NTSC 4.43 w/pedestal
  // 8: PAL BGHID w/o pedestal
  // 9: PAL N w/pedestal
  // A: PAL M w/o pedestal
  // B: PAL M w/pedestal
  // C: PAL combination N
  // D: PAL combination N w/pedestal
  // E-F: [Not valid]

`define ADV7185_REGISTER_0 {`INPUT_MODE, `INPUT_SELECT}
```

```
////////////////////////////////////////////////////////////////////////////
// Register 1
////////////////////////////////////////////////////////////////////////////

`define VIDEO_QUALITY                   2'h0
  // 0: Broadcast quality
  // 1: TV quality
  // 2: VCR quality
  // 3: Surveillance quality
`define SQUARE_PIXEL_IN_MODE            1'b0
  // 0: Normal mode
  // 1: Square pixel mode
`define DIFFERENTIAL_INPUT              1'b0
  // 0: Single-ended inputs
  // 1: Differential inputs
`define FOUR_TIMES_SAMPLING             1'b0
  // 0: Standard sampling rate
  // 1: 4x sampling rate (NTSC only)
`define BETACAM                         1'b0
  // 0: Standard video input
  // 1: Betacam video input
`define AUTOMATIC_STARTUP_ENABLE            1'b1
  // 0: Change of input triggers reacquire
  // 1: Change of input does not trigger reacquire

`define ADV7185_REGISTER_1 {`AUTOMATIC_STARTUP_ENABLE, 1'b0,
`BETACAM, `FOUR_TIMES_SAMPLING, `DIFFERENTIAL_INPUT,
`SQUARE_PIXEL_IN_MODE, `VIDEO_QUALITY}
```

```
////////////////////////////////////////////////////////////////////////////
// Register 2
////////////////////////////////////////////////////////////////////////////

`define Y_PEAKING_FILTER                    3'h4
  // 0: Composite =  4.5dB,  s-video =  9.25dB
  // 1: Composite =  4.5dB,  s-video =  9.25dB
  // 2: Composite =  4.5dB,  s-video =  5.75dB
  // 3: Composite =  1.25dB, s-video =  3.3dB
  // 4: Composite =  0.0dB,  s-video =  0.0dB
  // 5: Composite = -1.25dB, s-video = -3.0dB
  // 6: Composite = -1.75dB, s-video = -8.0dB
  // 7: Composite = -3.0dB,  s-video = -8.0dB
`define CORING                              2'h0
  // 0: No coring
  // 1: Truncate if Y < black+8
  // 2: Truncate if Y < black+16
  // 3: Truncate if Y < black+32

`define ADV7185_REGISTER_2 {3'b000, `CORING, `Y_PEAKING_FILTER}

////////////////////////////////////////////////////////////////////////////
// Register 3
////////////////////////////////////////////////////////////////////////////

`define INTERFACE_SELECT                    2'h0
  // 0: Philips-compatible
  // 1: Broktree API A-compatible
  // 2: Broktree API B-compatible
```

```
  // 3: [Not valid]
`define OUTPUT_FORMAT                   4'h0
  // 0: 10-bit @ LLC, 4:2:2 CCIR656
  // 1: 20-bit @ LLC, 4:2:2 CCIR656
  // 2: 16-bit @ LLC, 4:2:2 CCIR656
  // 3: 8-bit @ LLC, 4:2:2 CCIR656
  // 4: 12-bit @ LLC, 4:1:1
  // 5-F: [Not valid]
  // (Note that the 6.111 labkit hardware provides only a 10-bit interface to
  // the ADV7185.)
`define TRISTATE_OUTPUT_DRIVERS         1'b0
  // 0: Drivers tristated when ~OE is high
  // 1: Drivers always tristated
`define VBI_ENABLE                      1'b0
  // 0: Decode lines during vertical blanking interval
  // 1: Decode only active video regions

`define ADV7185_REGISTER_3 {`VBI_ENABLE, `TRISTATE_OUTPUT_DRIVERS,
`OUTPUT_FORMAT, `INTERFACE_SELECT}

////////////////////////////////////////////////////////////////////////////
// Register 4
////////////////////////////////////////////////////////////////////////////

`define OUTPUT_DATA_RANGE               1'b0
  // 0: Output values restricted to CCIR-compliant range
  // 1: Use full output range
`define BT656_TYPE                      1'b0
  // 0: BT656-3-compatible
```

```
  // 1: BT656-4-compatible

`define ADV7185_REGISTER_4 {`BT656_TYPE, 3'b000, 3'b110,
`OUTPUT_DATA_RANGE}

////////////////////////////////////////////////////////////////////////////
// Register 5
////////////////////////////////////////////////////////////////////////////


`define GENERAL_PURPOSE_OUTPUTS              4'b0000
`define GPO_0_1_ENABLE                       1'b0
  // 0: General purpose outputs 0 and 1 tristated
  // 1: General purpose outputs 0 and 1 enabled
`define GPO_2_3_ENABLE                       1'b0
  // 0: General purpose outputs 2 and 3 tristated
  // 1: General purpose outputs 2 and 3 enabled
`define BLANK_CHROMA_IN_VBI                  1'b1
  // 0: Chroma decoded and output during vertical blanking
  // 1: Chroma blanked during vertical blanking
`define HLOCK_ENABLE                         1'b0
  // 0: GPO 0 is a general purpose output
  // 1: GPO 0 shows HLOCK status

`define ADV7185_REGISTER_5 {`HLOCK_ENABLE, `BLANK_CHROMA_IN_VBI,
`GPO_2_3_ENABLE, `GPO_0_1_ENABLE, `GENERAL_PURPOSE_OUTPUTS}

////////////////////////////////////////////////////////////////////////////
// Register 7
```

```
////////////////////////////////////////////////////////////////////////////

`define FIFO_FLAG_MARGIN                    5'h10
  // Sets the locations where FIFO almost-full and almost-empty flags are set
`define FIFO_RESET                          1'b0
  // 0: Normal operation
  // 1: Reset FIFO. This bit is automatically cleared
`define AUTOMATIC_FIFO_RESET                1'b0
  // 0: No automatic reset
  // 1: FIFO is autmatically reset at the end of each video field
`define FIFO_FLAG_SELF_TIME                 1'b1
  // 0: FIFO flags are synchronized to CLKIN
  // 1: FIFO flags are synchronized to internal 27MHz clock

`define ADV7185_REGISTER_7 {`FIFO_FLAG_SELF_TIME,
`AUTOMATIC_FIFO_RESET, `FIFO_RESET, `FIFO_FLAG_MARGIN}

////////////////////////////////////////////////////////////////////////////
// Register 8
////////////////////////////////////////////////////////////////////////////

`define INPUT_CONTRAST_ADJUST               8'h80

`define ADV7185_REGISTER_8 {`INPUT_CONTRAST_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register 9
////////////////////////////////////////////////////////////////////////////
```

```
`define INPUT_SATURATION_ADJUST                 8'h8C

`define ADV7185_REGISTER_9 {`INPUT_SATURATION_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register A
////////////////////////////////////////////////////////////////////////////

`define INPUT_BRIGHTNESS_ADJUST                 8'h00

`define ADV7185_REGISTER_A {`INPUT_BRIGHTNESS_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register B
////////////////////////////////////////////////////////////////////////////

`define INPUT_HUE_ADJUST                    8'h00

`define ADV7185_REGISTER_B {`INPUT_HUE_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register C
////////////////////////////////////////////////////////////////////////////

`define DEFAULT_VALUE_ENABLE                1'b0
  // 0: Use programmed Y, Cr, and Cb values
  // 1: Use default values
`define DEFAULT_VALUE_AUTOMATIC_ENABLE              1'b0
  // 0: Use programmed Y, Cr, and Cb values
```

```
  // 1: Use default values if lock is lost
`define DEFAULT_Y_VALUE                 6'h0C
  // Default Y value

`define ADV7185_REGISTER_C {`DEFAULT_Y_VALUE,
`DEFAULT_VALUE_AUTOMATIC_ENABLE, `DEFAULT_VALUE_ENABLE}

////////////////////////////////////////////////////////////////////////////
// Register D
////////////////////////////////////////////////////////////////////////////

`define DEFAULT_CR_VALUE                4'h8
  // Most-significant four bits of default Cr value
`define DEFAULT_CB_VALUE                4'h8
  // Most-significant four bits of default Cb value

`define ADV7185_REGISTER_D {`DEFAULT_CB_VALUE, `DEFAULT_CR_VALUE}

////////////////////////////////////////////////////////////////////////////
// Register E
////////////////////////////////////////////////////////////////////////////

`define TEMPORAL_DECIMATION_ENABLE      1'b0
  // 0: Disable
  // 1: Enable
`define TEMPORAL_DECIMATION_CONTROL     2'h0
  // 0: Supress frames, start with even field
  // 1: Supress frames, start with odd field
  // 2: Supress even fields only
```

```
  // 3: Supress odd fields only
`define TEMPORAL_DECIMATION_RATE              4'h0
  // 0-F: Number of fields/frames to skip

`define ADV7185_REGISTER_E {1'b0, `TEMPORAL_DECIMATION_RATE,
`TEMPORAL_DECIMATION_CONTROL, `TEMPORAL_DECIMATION_ENABLE}

////////////////////////////////////////////////////////////////////////////
// Register F
////////////////////////////////////////////////////////////////////////////

`define POWER_SAVE_CONTROL                2'h0
  // 0: Full operation
  // 1: CVBS only
  // 2: Digital only
  // 3: Power save mode
`define POWER_DOWN_SOURCE_PRIORITY        1'b0
  // 0: Power-down pin has priority
  // 1: Power-down control bit has priority
`define POWER_DOWN_REFERENCE              1'b0
  // 0: Reference is functional
  // 1: Reference is powered down
`define POWER_DOWN_LLC_GENERATOR          1'b0
  // 0: LLC generator is functional
  // 1: LLC generator is powered down
`define POWER_DOWN_CHIP                   1'b0
  // 0: Chip is functional
  // 1: Input pads disabled and clocks stopped
`define TIMING_REACQUIRE                  1'b0
```

```
  // 0: Normal operation
  // 1: Reacquire video signal (bit will automatically reset)
`define RESET_CHIP                          1'b0
  // 0: Normal operation
  // 1: Reset digital core and I2C interface (bit will automatically reset)

`define ADV7185_REGISTER_F {`RESET_CHIP, `TIMING_REACQUIRE,
`POWER_DOWN_CHIP, `POWER_DOWN_LLC_GENERATOR,
`POWER_DOWN_REFERENCE, `POWER_DOWN_SOURCE_PRIORITY,
`POWER_SAVE_CONTROL}

////////////////////////////////////////////////////////////////////////////
// Register 33
////////////////////////////////////////////////////////////////////////////

`define PEAK_WHITE_UPDATE                   1'b1
  // 0: Update gain once per line
  // 1: Update gain once per field
`define AVERAGE_BIRIGHTNESS_LINES           1'b1
  // 0: Use lines 33 to 310
  // 1: Use lines 33 to 270
`define MAXIMUM_IRE                         3'h0
  // 0: PAL: 133, NTSC: 122
  // 1: PAL: 125, NTSC: 115
  // 2: PAL: 120, NTSC: 110
  // 3: PAL: 115, NTSC: 105
  // 4: PAL: 110, NTSC: 100
  // 5: PAL: 105, NTSC: 100
  // 6-7: PAL: 100, NTSC: 100
```

```
`define COLOR_KILL                    1'b1
  // 0: Disable color kill
  // 1: Enable color kill

`define ADV7185_REGISTER_33 {1'b1, `COLOR_KILL, 1'b1, `MAXIMUM_IRE,
`AVERAGE_BIRIGHTNESS_LINES, `PEAK_WHITE_UPDATE}

`define ADV7185_REGISTER_10 8'h00
`define ADV7185_REGISTER_11 8'h00
`define ADV7185_REGISTER_12 8'h00
`define ADV7185_REGISTER_13 8'h45
`define ADV7185_REGISTER_14 8'h18
`define ADV7185_REGISTER_15 8'h60
`define ADV7185_REGISTER_16 8'h00
`define ADV7185_REGISTER_17 8'h01
`define ADV7185_REGISTER_18 8'h00
`define ADV7185_REGISTER_19 8'h10
`define ADV7185_REGISTER_1A 8'h10
`define ADV7185_REGISTER_1B 8'hF0
`define ADV7185_REGISTER_1C 8'h16
`define ADV7185_REGISTER_1D 8'h01
`define ADV7185_REGISTER_1E 8'h00
`define ADV7185_REGISTER_1F 8'h3D
`define ADV7185_REGISTER_20 8'hD0
`define ADV7185_REGISTER_21 8'h09
`define ADV7185_REGISTER_22 8'h8C
`define ADV7185_REGISTER_23 8'hE2
`define ADV7185_REGISTER_24 8'h1F
`define ADV7185_REGISTER_25 8'h07
```

```
`define ADV7185_REGISTER_26 8'hC2
`define ADV7185_REGISTER_27 8'h58
`define ADV7185_REGISTER_28 8'h3C
`define ADV7185_REGISTER_29 8'h00
`define ADV7185_REGISTER_2A 8'h00
`define ADV7185_REGISTER_2B 8'hA0
`define ADV7185_REGISTER_2C 8'hCE
`define ADV7185_REGISTER_2D 8'hF0
`define ADV7185_REGISTER_2E 8'h00
`define ADV7185_REGISTER_2F 8'hF0
`define ADV7185_REGISTER_30 8'h00
`define ADV7185_REGISTER_31 8'h70
`define ADV7185_REGISTER_32 8'h00
`define ADV7185_REGISTER_34 8'h0F
`define ADV7185_REGISTER_35 8'h01
`define ADV7185_REGISTER_36 8'h00
`define ADV7185_REGISTER_37 8'h00
`define ADV7185_REGISTER_38 8'h00
`define ADV7185_REGISTER_39 8'h00
`define ADV7185_REGISTER_3A 8'h00
`define ADV7185_REGISTER_3B 8'h00

`define ADV7185_REGISTER_44 8'h41
`define ADV7185_REGISTER_45 8'hBB

`define ADV7185_REGISTER_F1 8'hEF
`define ADV7185_REGISTER_F2 8'h80
```

```
module adv7185init (reset, clock_27mhz, source, tv_in_reset_b,
                    tv_in_i2c_clock, tv_in_i2c_data);

  input reset;
  input clock_27mhz;
  output tv_in_reset_b; // Reset signal to ADV7185
  output tv_in_i2c_clock; // I2C clock output to ADV7185
  output tv_in_i2c_data; // I2C data line to ADV7185
  input source; // 0: composite, 1: s-video

  initial begin
      $display("ADV7185 Initialization values:");
      $display("  Register 0:  0x%X", `ADV7185_REGISTER_0);
      $display("  Register 1:  0x%X", `ADV7185_REGISTER_1);
      $display("  Register 2:  0x%X", `ADV7185_REGISTER_2);
      $display("  Register 3:  0x%X", `ADV7185_REGISTER_3);
      $display("  Register 4:  0x%X", `ADV7185_REGISTER_4);
      $display("  Register 5:  0x%X", `ADV7185_REGISTER_5);
      $display("  Register 7:  0x%X", `ADV7185_REGISTER_7);
      $display("  Register 8:  0x%X", `ADV7185_REGISTER_8);
      $display("  Register 9:  0x%X", `ADV7185_REGISTER_9);
      $display("  Register A:  0x%X", `ADV7185_REGISTER_A);
      $display("  Register B:  0x%X", `ADV7185_REGISTER_B);
      $display("  Register C:  0x%X", `ADV7185_REGISTER_C);
      $display("  Register D:  0x%X", `ADV7185_REGISTER_D);
      $display("  Register E:  0x%X", `ADV7185_REGISTER_E);
      $display("  Register F:  0x%X", `ADV7185_REGISTER_F);
      $display("  Register 33: 0x%X", `ADV7185_REGISTER_33);
  end
```

```
//
// Generate a 1MHz for the I2C driver (resulting I2C clock rate is 250kHz)
//

reg [7:0] clk_div_count, reset_count;
reg clock_slow;
wire reset_slow;

initial
        begin
 clk_div_count <= 8'h00;
 // synthesis attribute init of clk_div_count is "00";
 clock_slow <= 1'b0;
 // synthesis attribute init of clock_slow is "0";
        end

always @(posedge clock_27mhz)
        if (clk_div_count == 26)
        begin
   clock_slow <= ~clock_slow;
   clk_div_count <= 0;
        end
        else
        clk_div_count <= clk_div_count+1;

always @(posedge clock_27mhz)
        if (reset)
        reset_count <= 100;
```

```
        else
        reset_count <= (reset_count==0) ? 0 : reset_count-1;

assign reset_slow = reset_count != 0;

//
// I2C driver
//

reg load;
reg [7:0] data;
wire ack, idle;

i2c i2c(.reset(reset_slow), .clock4x(clock_slow), .data(data), .load(load),
        .ack(ack), .idle(idle), .scl(tv_in_i2c_clock),
        .sda(tv_in_i2c_data));

//
// State machine
//

reg [7:0] state;
reg tv_in_reset_b;
reg old_source;

always @(posedge clock_slow)
        if (reset_slow)
 begin
        state <= 0;
```

```
      load <= 0;
      tv_in_reset_b <= 0;
      old_source <= 0;
end
      else
case (state)
  8'h00:
      begin
      // Assert reset
      load <= 1'b0;
      tv_in_reset_b <= 1'b0;
      if (!ack)
        state <= state+1;
      end
  8'h01:
      state <= state+1;
  8'h02:
      begin
      // Release reset
      tv_in_reset_b <= 1'b1;
      state <= state+1;
                        end
  8'h03:
      begin
      // Send ADV7185 address
      data <= 8'h8A;
      load <= 1'b1;
      if (ack)
        state <= state+1;
```

```
    end
8'h04:
    begin
    // Send subaddress of first register
    data <= 8'h00;
    if (ack)
      state <= state+1;
    end
8'h05:
    begin
    // Write to register 0
    data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
    if (ack)
      state <= state+1;
    end
8'h06:
    begin
    // Write to register 1
    data <= `ADV7185_REGISTER_1;
    if (ack)
      state <= state+1;
    end
8'h07:
    begin
    // Write to register 2
    data <= `ADV7185_REGISTER_2;
    if (ack)
      state <= state+1;
    end
```

```
8'h08:
    begin
    // Write to register 3
    data <= `ADV7185_REGISTER_3;
    if (ack)
      state <= state+1;
    end
8'h09:
    begin
    // Write to register 4
    data <= `ADV7185_REGISTER_4;
    if (ack)
      state <= state+1;
    end
8'h0A:
    begin
    // Write to register 5
    data <= `ADV7185_REGISTER_5;
    if (ack)
      state <= state+1;
    end
8'h0B:
    begin
    // Write to register 6
    data <= 8'h00; // Reserved register, write all zeros
    if (ack)
      state <= state+1;
    end
8'h0C:
```

```
    begin
    // Write to register 7
    data <= `ADV7185_REGISTER_7;
    if (ack)
      state <= state+1;
    end
8'h0D:
    begin
    // Write to register 8
    data <= `ADV7185_REGISTER_8;
    if (ack)
      state <= state+1;
    end
8'h0E:
    begin
    // Write to register 9
    data <= `ADV7185_REGISTER_9;
    if (ack)
      state <= state+1;
    end
8'h0F: begin
    // Write to register A
    data <= `ADV7185_REGISTER_A;
    if (ack)
    state <= state+1;
end
8'h10:
    begin
    // Write to register B
```

```
    data <= `ADV7185_REGISTER_B;
    if (ack)
      state <= state+1;
    end
8'h11:
    begin
    // Write to register C
    data <= `ADV7185_REGISTER_C;
    if (ack)
      state <= state+1;
    end
8'h12:
    begin
    // Write to register D
    data <= `ADV7185_REGISTER_D;
    if (ack)
      state <= state+1;
    end
8'h13:
    begin
    // Write to register E
    data <= `ADV7185_REGISTER_E;
    if (ack)
      state <= state+1;
    end
8'h14:
    begin
    // Write to register F
    data <= `ADV7185_REGISTER_F;
```

```
    if (ack)
      state <= state+1;
    end
8'h15:
    begin
    // Wait for I2C transmitter to finish
    load <= 1'b0;
    if (idle)
      state <= state+1;
    end
8'h16:
    begin
    // Write address
    data <= 8'h8A;
    load <= 1'b1;
    if (ack)
      state <= state+1;
    end
8'h17:
    begin
    data <= 8'h33;
    if (ack)
      state <= state+1;
    end
8'h18:
    begin
    data <= `ADV7185_REGISTER_33;
    if (ack)
      state <= state+1;
```

```
    end
8'h19:
    begin
    load <= 1'b0;
    if (idle)
      state <= state+1;
    end

8'h1A: begin
    data <= 8'h8A;
    load <= 1'b1;
    if (ack)
    state <= state+1;
end
8'h1B:
    begin
    data <= 8'h33;
    if (ack)
      state <= state+1;
    end
8'h1C:
    begin
    load <= 1'b0;
    if (idle)
      state <= state+1;
    end
8'h1D:
    begin
    load <= 1'b1;
```

```
    data <= 8'h8B;
    if (ack)
      state <= state+1;
    end
8'h1E:
    begin
    data <= 8'hFF;
    if (ack)
      state <= state+1;
    end
8'h1F:
    begin
    load <= 1'b0;
    if (idle)
      state <= state+1;
    end
8'h20:
    begin
    // Idle
    if (old_source != source) state <= state+1;
    old_source <= source;
    end
8'h21: begin
    // Send ADV7185 address
    data <= 8'h8A;
    load <= 1'b1;
    if (ack) state <= state+1;
end
8'h22: begin
```

```
        // Send subaddress of register 0
        data <= 8'h00;
        if (ack) state <= state+1;
    end
    8'h23: begin
        // Write to register 0
        data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
        if (ack) state <= state+1;
    end
    8'h24: begin
        // Wait for I2C transmitter to finish
        load <= 1'b0;
        if (idle) state <= 8'h20;
    end
        endcase

endmodule

// i2c module for use with the ADV7185

module i2c (reset, clock4x, data, load, idle, ack, scl, sda);

  input reset;
  input clock4x;
  input [7:0] data;
  input load;
  output ack;
  output idle;
  output scl;
```

```
output sda;

reg [7:0] ldata;
reg ack, idle;
reg scl;
reg sdai;

reg [7:0] state;

assign sda = sdai ? 1'bZ : 1'b0;

always @(posedge clock4x)
        if (reset)
        begin
   state <= 0;
   ack <= 0;
        end
        else
        case (state)
  8'h00: // idle
        begin
        scl <= 1'b1;
        sdai <= 1'b1;
        ack <= 1'b0;
        idle <= 1'b1;
        if (load)
         begin
                ldata <= data;
                ack <= 1'b1;
```

```
                state <= state+1;
        end
        end
8'h01: // Start
        begin
        ack <= 1'b0;
        idle <= 1'b0;
        sdai <= 1'b0;
        state <= state+1;
        end
8'h02:
        begin
        scl <= 1'b0;
        state <= state+1;
        end
8'h03: // Send bit 7
        begin
        ack <= 1'b0;
        sdai <= ldata[7];
        state <= state+1;
        end
8'h04:
        begin
        scl <= 1'b1;
        state <= state+1;
        end
8'h05:
        begin
        state <= state+1;
```

```
    end
8'h06:
    begin
    scl <= 1'b0;
    state <= state+1;
    end
8'h07:
    begin
    sdai <= ldata[6];
    state <= state+1;
    end
8'h08:
    begin
    scl <= 1'b1;
    state <= state+1;
    end
8'h09:
    begin
    state <= state+1;
    end
8'h0A:
    begin
    scl <= 1'b0;
    state <= state+1;
    end
8'h0B:
    begin
    sdai <= ldata[5];
    state <= state+1;
```

```
    end
8'h0C:
    begin
    scl <= 1'b1;
    state <= state+1;
    end
8'h0D:
    begin
    state <= state+1;
    end
8'h0E:
    begin
    scl <= 1'b0;
    state <= state+1;
    end
8'h0F:
    begin
    sdai <= ldata[4];
    state <= state+1;
    end
8'h10:
    begin
    scl <= 1'b1;
    state <= state+1;
    end
8'h11:
    begin
    state <= state+1;
    end
```

```
8'h12:
    begin
    scl <= 1'b0;
    state <= state+1;
    end
8'h13:
    begin
    sdai <= ldata[3];
    state <= state+1;
    end
8'h14:
    begin
    scl <= 1'b1;
    state <= state+1;
    end
8'h15:
    begin
    state <= state+1;
    end
8'h16:
    begin
    scl <= 1'b0;
    state <= state+1;
    end
8'h17:
    begin
    sdai <= ldata[2];
    state <= state+1;
    end
```

```
8'h18:
    begin
    scl <= 1'b1;
    state <= state+1;
    end
8'h19:
    begin
    state <= state+1;
    end
8'h1A:
    begin
    scl <= 1'b0;
    state <= state+1;
    end
8'h1B:
    begin
    sdai <= ldata[1];
    state <= state+1;
    end
8'h1C:
    begin
    scl <= 1'b1;
    state <= state+1;
    end
8'h1D:
    begin
    state <= state+1;
    end
8'h1E:
```

```
    begin
    scl <= 1'b0;
    state <= state+1;
    end
8'h1F:
    begin
    sdai <= ldata[0];
    state <= state+1;
    end
8'h20:
    begin
    scl <= 1'b1;
    state <= state+1;
    end
8'h21:
    begin
    state <= state+1;
    end
8'h22:
    begin
    scl <= 1'b0;
    state <= state+1;
    end
8'h23: // Acknowledge bit
    begin
    state <= state+1;
    end
8'h24:
    begin
```

```
    scl <= 1'b1;
    state <= state+1;
    end
8'h25:
    begin
    state <= state+1;
    end
8'h26:
    begin
    scl <= 1'b0;
    if (load)
     begin
            ldata <= data;
            ack <= 1'b1;
            state <= 3;
     end
    else
     state <= state+1;
    end
8'h27:
    begin
    sdai <= 1'b0;
    state <= state+1;
    end
8'h28:
    begin
    scl <= 1'b1;
    state <= state+1;
    end
```

```
    8'h29:
        begin
        sdai <= 1'b1;
        state <= 0;
        end
        endcase

endmodule
```

ntsc2zbt.v

```
//
// File:   ntsc2zbt.v
// Date:   27-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Example for MIT 6.111 labkit showing how to prepare NTSC data
// (from Javier's decoder) to be loaded into the ZBT RAM for video
// display.
//
// The ZBT memory is 36 bits wide; we only use 32 bits of this, to
// store 4 bytes of black-and-white intensity data from the NTSC
// video input.
//
// Bug fix: Jonathan P. Mailoa <jpmailoa@mit.edu>
// Date   : 11-May-09  // gph mod 11/3/2011
//
//
// Bug due to memory management will be fixed. It happens because
```

```
// the memory addressing protocol is off between ntsc2zbt.v and
// vram_display.v. There are 2 solutions:
// -. Fix the memory addressing in this module (neat addressing protocol)
//        and do memory forecast in vram_display module.
// -. Do nothing in this module and do memory forecast in vram_display
//        module (different forecast count) while cutting off reading from
//        address(0,0,0).
//
// Bug in this module causes 4 pixel on the rightmost side of the camera
// to be stored in the address that belongs to the leftmost side of the
// screen.
//
// In this example, the second method is used. NOTICE will be provided
// on the crucial source of the bug.
//
//////////////////////////////////////////////////////////////////////////////
// Prepare data and address values to fill ZBT memory with NTSC data

module ntsc_to_zbt(clk, vclk, fvh, dv, din, ntsc_addr, ntsc_data, ntsc_we, sw);

  input      clk;    // system clock
  input      vclk;    // video clock from camera
  input [2:0]      fvh;
  input      dv;
  input [7:0]      din;
  output [18:0] ntsc_addr;
  output [35:0] ntsc_data;
  output      ntsc_we;    // write enable for NTSC data
  input      sw;          // switch which determines mode (for debugging)
```

```
parameter      COL_START = 10'd30;
parameter      ROW_START = 10'd30;

// here put the luminance data from the ntsc decoder into the ram
// this is for 1024 * 788 XGA display

reg [9:0]      col = 0;
reg [9:0]      row = 0;
reg [7:0]      vdata = 0;
reg             vwe;
reg             old_dv;
reg             old_frame;    // frames are even / odd interlaced
reg             even_odd;    // decode interlaced frame to this wire

wire      frame = fvh[2];
wire      frame_edge = frame & ~old_frame;

always @ (posedge vclk) //LLC1 is reference
       begin
 old_dv <= dv;
 vwe <= dv && !fvh[2] & ~old_dv; // if data valid, write it
 old_frame <= frame;
 even_odd = frame_edge ? ~even_odd : even_odd;

 if (!fvh[2])
   begin
       col <= fvh[0] ? COL_START :
                (!fvh[2] && !fvh[1] && dv && (col < 1024)) ? col + 1 : col;
```

```
        row <= fvh[1] ? ROW_START :
                (!fvh[2] && fvh[0] && (row < 768)) ? row + 1 : row;
        vdata <= (dv && !fvh[2]) ? din : vdata;
    end
        end

// synchronize with system clock

reg [9:0] x[1:0],y[1:0];
reg [7:0] data[1:0];
reg       we[1:0];
reg       eo[1:0];

always @(posedge clk)
        begin
 {x[1],x[0]} <= {x[0],col};
 {y[1],y[0]} <= {y[0],row};
 {data[1],data[0]} <= {data[0],vdata};
 {we[1],we[0]} <= {we[0],vwe};
 {eo[1],eo[0]} <= {eo[0],even_odd};
        end

// edge detection on write enable signal

reg old_we;
wire we_edge = we[1] & ~old_we;
always @(posedge clk) old_we <= we[1];

// shift each set of four bytes into a large register for the ZBT
```

```
reg [31:0] mydata;
always @(posedge clk)
	if (we_edge)
	mydata <= { mydata[23:0], data[1] };

// NOTICE : Here we have put 4 pixel delay on mydata. For example, when:
// (x[1], y[1]) = (60, 80) and eo[1] = 0, then:
// mydata[31:0] = ( pixel(56,160), pixel(57,160), pixel(58,160), pixel(59,160) )
// This is the root of the original addressing bug.


// NOTICE : Notice that we have decided to store mydata, which
//		contains pixel(56,160) to pixel(59,160) in address
//		(0, 160 (10 bits), 60 >> 2 = 15 (8 bits)).
//
//		This protocol is dangerous, because it means
//		pixel(0,0) to pixel(3,0) is NOT stored in address
//		(0, 0 (10 bits), 0 (8 bits)) but is rather stored
//		in address (0, 0 (10 bits), 4 >> 2 = 1 (8 bits)). This
//		calculation ignores COL_START & ROW_START.
//
//		4 pixels from the right side of the camera input will
//		be stored in address corresponding to x = 0.
//
//		To fix, delay col & row by 4 clock cycles.
//		Delay other signals as well.

reg [39:0] x_delay;
```

```
reg [39:0] y_delay;
reg [3:0] we_delay;
reg [3:0] eo_delay;

always @ (posedge clk)
begin
        x_delay <= {x_delay[29:0], x[1]};
        y_delay <= {y_delay[29:0], y[1]};
        we_delay <= {we_delay[2:0], we[1]};
        eo_delay <= {eo_delay[2:0], eo[1]};
end

// compute address to store data in
wire [8:0] y_addr = y_delay[38:30];
 wire [9:0] x_addr = x_delay[39:30];

wire [18:0] myaddr = {1'b0, y_addr[8:0], eo_delay[3], x_addr[9:2]};

// Now address (0,0,0) contains pixel data(0,0) etc.



// alternate (256x192) image data and address
wire [31:0] mydata2 = {data[1],data[1],data[1],data[1]};
wire [18:0] myaddr2 = {1'b0, y_addr[8:0], eo_delay[3], x_addr[7:0]};

// update the output address and data only when four bytes ready

reg [18:0] ntsc_addr;
reg [35:0] ntsc_data;
```

```
  wire          ntsc_we = sw ? we_edge : (we_edge & (x_delay[31:30]==2'b00));

  always @(posedge clk)
        if ( ntsc_we )
        begin
    ntsc_addr <= sw ? myaddr2 : myaddr;    // normal and expanded modes
    ntsc_data <= sw ? {4'b0,mydata2} : {4'b0,mydata};
        end

endmodule // ntsc_to_zbt
```

display_16hex.v

```
////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Hex display driver
//
//
// File:   display_16hex.v
// Date:   24-Sep-05
//
// Created: April 27, 2004
// Author: Nathan Ickes
//
// This module drives the labkit hex displays and shows the value of
// 8 bytes (16 hex digits) on the displays.
//
// 24-Sep-05 Ike: updated to use new reset-once state machine, remove clear
```

```
// 02-Nov-05 Ike: updated to make it completely synchronous
//
// Inputs:
//
//   reset        - active high
//   clock_27mhz - the synchronous clock
//   data         - 64 bits; each 4 bits gives a hex digit
//
// Outputs:
//
//        disp_*          - display lines used in the 6.111 labkit (rev 003 & 004)
//
//////////////////////////////////////////////////////////////////////////////////

module display_16hex (reset, clock_27mhz, data_in,
          disp_blank, disp_clock, disp_rs, disp_ce_b,
          disp_reset_b, disp_data_out);

  input reset, clock_27mhz;          // clock and reset (active high reset)
  input [63:0] data_in;         // 16 hex nibbles to display

  output disp_blank, disp_clock, disp_data_out, disp_rs, disp_ce_b,
    disp_reset_b;

  reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;

  //////////////////////////////////////////////////////////////////////////////////
  //
  // Display Clock
```

```
   //
   // Generate a 500kHz clock for driving the displays.
   //
   ////////////////////////////////////////////////////////////////////////

   reg [5:0] count;
   reg [7:0] reset_count;
//   reg          old_clock;
   wire         dreset;
   wire         clock = (count<27) ? 0 : 1;

   always @(posedge clock_27mhz)
         begin
    count <= reset ? 0 : (count==53 ? 0 : count+1);
    reset_count <= reset ? 100 : ((reset_count==0) ? 0 : reset_count-1);
//    old_clock <= clock;
         end

   assign dreset = (reset_count != 0);
   assign disp_clock = ~clock;
   wire   clock_tick = ((count==27) ? 1 : 0);
//   wire   clock_tick = clock & ~old_clock;

   ////////////////////////////////////////////////////////////////////////
   //
   // Display State Machine
   //
   ////////////////////////////////////////////////////////////////////////
```

```
reg [7:0] state;       // FSM state
reg [9:0] dot_index;          // index to current dot being clocked out
reg [31:0] control;           // control register
reg [3:0] char_index;    // index of current character
reg [39:0] dots;       // dots for a single digit
reg [3:0] nibble;      // hex nibble of current character
reg [63:0] data;

assign disp_blank = 1'b0; // low <= not blanked

always @(posedge clock_27mhz)
      if (clock_tick)
      begin
  if (dreset)
      begin
      state <= 0;
      dot_index <= 0;
      control <= 32'h7F7F7F7F;
      end
  else
      casex (state)
      8'h00:
       begin
            // Reset displays
            disp_data_out <= 1'b0;
            disp_rs <= 1'b0; // dot register
            disp_ce_b <= 1'b1;
            disp_reset_b <= 1'b0;
            dot_index <= 0;
```

```
        state <= state+1;
 end

8'h01:
 begin
        // End reset
        disp_reset_b <= 1'b1;
        state <= state+1;
 end

8'h02:
 begin
        // Initialize dot register (set all dots to zero)
        disp_ce_b <= 1'b0;
        disp_data_out <= 1'b0; // dot_index[0];
        if (dot_index == 639)
        state <= state+1;
        else
        dot_index <= dot_index+1;
 end

8'h03:
 begin
        // Latch dot data
        disp_ce_b <= 1'b1;
        dot_index <= 31;      // re-purpose to init ctrl reg
        state <= state+1;
 end
```

```
8'h04:
 begin
        // Setup the control register
        disp_rs <= 1'b1; // Select the control register
        disp_ce_b <= 1'b0;
        disp_data_out <= control[31];
        control <= {control[30:0], 1'b0};    // shift left
        if (dot_index == 0)
        state <= state+1;
        else
        dot_index <= dot_index-1;
 end

8'h05:
 begin
        // Latch the control register data / dot data
        disp_ce_b <= 1'b1;
        dot_index <= 39;       // init for single char
        char_index <= 15;     // start with MS char
        data <= data_in;
        state <= state+1;
 end

8'h06:
 begin
        // Load the user's dot data into the dot reg, char by char
        disp_rs <= 1'b0;                  // Select the dot register
        disp_ce_b <= 1'b0;
        disp_data_out <= dots[dot_index]; // dot data from msb
```

```
            if (dot_index == 0)
            if (char_index == 0)
            state <= 5;              // all done, latch data
            else
            begin
              char_index <= char_index - 1;    // goto next char
              data <= data_in;
              dot_index <= 39;
            end
            else
            dot_index <= dot_index-1;    // else loop thru all dots
         end

      endcase // casex(state)
      end

always @ (data or char_index)
      case (char_index)
      4'h0:         nibble <= data[3:0];
      4'h1:         nibble <= data[7:4];
      4'h2:         nibble <= data[11:8];
      4'h3:         nibble <= data[15:12];
      4'h4:         nibble <= data[19:16];
      4'h5:         nibble <= data[23:20];
      4'h6:         nibble <= data[27:24];
      4'h7:         nibble <= data[31:28];
      4'h8:         nibble <= data[35:32];
      4'h9:         nibble <= data[39:36];
      4'hA:         nibble <= data[43:40];
```

```
        4'hB:       nibble <= data[47:44];
        4'hC:       nibble <= data[51:48];
        4'hD:       nibble <= data[55:52];
        4'hE:       nibble <= data[59:56];
        4'hF:       nibble <= data[63:60];
        endcase

    always @(nibble)
        case (nibble)
        4'h0: dots <= 40'b00111110_01010001_01001001_01000101_00111110;
        4'h1: dots <= 40'b00000000_01000010_01111111_01000000_00000000;
        4'h2: dots <= 40'b01100010_01010001_01001001_01001001_01000110;
        4'h3: dots <= 40'b00100010_01000001_01001001_01001001_00110110;
        4'h4: dots <= 40'b00011000_00010100_00010010_01111111_00010000;
        4'h5: dots <= 40'b00100111_01000101_01000101_01000101_00111001;
        4'h6: dots <= 40'b00111100_01001010_01001001_01001001_00110000;
        4'h7: dots <= 40'b00000001_01110001_00001001_00000101_00000011;
        4'h8: dots <= 40'b00110110_01001001_01001001_01001001_00110110;
        4'h9: dots <= 40'b00000110_01001001_01001001_00101001_00011110;
        4'hA: dots <= 40'b01111110_00001001_00001001_00001001_01111110;
        4'hB: dots <= 40'b01111111_01001001_01001001_01001001_00110110;
        4'hC: dots <= 40'b00111110_01000001_01000001_01000001_00100010;
        4'hD: dots <= 40'b01111111_01000001_01000001_01000001_00111110;
        4'hE: dots <= 40'b01111111_01001001_01001001_01001001_01000001;
        4'hF: dots <= 40'b01111111_00001001_00001001_00001001_00000001;
        endcase

endmodule
```

debounce.v

```
////////////////////////////////////////////////////////////////////////////////
//
// Pushbutton Debounce Module
//
////////////////////////////////////////////////////////////////////////////////

//module debounce (reset, clk, noisy, clean);
//   input reset, clk, noisy;
//   output clean;
//
//   parameter NDELAY = 650000;
//   parameter NBITS = 20;
//
//   reg [NBITS-1:0] count;
//   reg xnew, clean;
//
//   always @(posedge clk)
//         if (reset) begin xnew <= noisy; clean <= noisy; count <= 0; end
//         else if (noisy != xnew) begin xnew <= noisy; count <= 0; end
//         else if (count == NDELAY) clean <= xnew;
//         else count <= count+1;
//
//endmodule

// Switch Debounce Module
// use your system clock for the clock input
// to produce a synchronous, debounced output
```

```
module debounce #(parameter DELAY=1000000)   // .01 sec with a 100Mhz clock
        (input reset, clk, noisy,
        output reg clean);

  reg [19:0] count;
  reg new;

  always @(posedge clk)
        if (reset)
        begin
    count <= 0;
    new <= noisy;
    clean <= noisy;
        end
        else if (noisy != new)
        begin
    new <= noisy;
    count <= 0;
        end
        else if (count == DELAY)
        clean <= new;
        else
        count <= count+1;

endmodule
```